The Code Review Comprehension Assessment for Language Models

Anonymous ACL submission

Abstract

State-of-the-art language models have demonstrated impressive code generation capabilities but struggle with real-world software engineering tasks like code reviewing, hindering practical use. Review comments are often implicit, ambiguous, and colloquial, requiring models to grasp both code and human intent. This challenge calls for evaluating language models' ability to bridge technical and conversational contexts. While existing work has employed the automated code refinement task to resolve these comments, current evaluation methods fall short, relying on metrics that provide limited insight into model failures and remain susceptible to training data contamination. To address these limitations, we introduce a novel evaluation benchmark **CodeReviewQA** that enables us to conduct fine-grained assessment of model capabilities and mitigate data contamination risks. In **CodeReviewQA**, we decompose the generation task in code refinement into three essential reasoning steps: change type recognition, change localisation, and solution identification. Each step is reformulated as multiple-choice questions with varied difficulty levels, enabling precise assessment of model capabilities while mitigating data contamination risks. Our comprehensive evaluation spans 65 recently released large language models on 900 manually curated, high-quality examples across nine programming languages. Our results show that CodeReviewQA is able to reveal model capability gaps in different reasoning tasks, and expose specific model weaknesses.[1]

1 Introduction

The proficiency of state-of-the-art large language models (LLMs) in code generation has garnered significant attention (Zhuo et al., 2024), demonstrating their ability to follow explicit instructions to author code. However, their competency in real-world software engineering environments remains limited (Pornprasit and Tantithamthavorn, 2024), particularly in collaborative tasks involving colloquial and complex forms of communication. A quintessential example is code reviewing, where review comments (Yang et al., 2023; Efstathiou and Spinellis, 2018) represent natural communication between developers with a shared mental model, often resulting in under-specified, ambiguous, and implicit expressions of intent. For example, this comment *"For all of the fuzz tests, does it make sense to have versions for 'len_prefixed' both 'true' and 'false'?"* is asking for an opinion, rather than giving explicit instructions.

As a result, the ability to resolve code review comments requires not only proficiency in understanding and generating code but also the ability to comprehend the communicative intent behind the code review in relation to the code submission it addresses. Therefore, we argue that assessing how LLMs resolve code review comments thus serves as a crucial testbed for their capability to understand and follow on implicit, conversational instructions in software development. Success in this domain would significantly advance automated software development assistance, potentially reducing developer workload and improving code quality.

To test models' capability in resolving code review comments, prior work has explored automated code refinement task using both neural models (Tufano et al., 2022; Thongtanunam et al., 2022) and LLMs (Guo et al., 2024; Pornprasit and Tantithamthavorn, 2024), which aims to revise code based on peer review comments. While efforts have advanced this direction, several critical challenges remain unaddressed. First, current automatic evaluation approaches rely heavily on metrics such as exact match and BLEU (Papineni et al., 2002), which merely capture surface-level token similarities without assessing deeper comprehension capabilities. Second, as these evaluation benchmarks typically use popular GitHub projects, they risk data contam-

[1] All code and data will be released upon acceptance.

ination from training data in LLMs (Sallou et al., 2024), potentially masking true model capabilities. As a result, there are no suitable evaluation benchmarks and approaches to assess LLMs' capabilities in code reviewing.

To address these challenges, we introduce a novel evaluation benchmark that enables comprehensive assessment of automated code refinement capabilities. Our benchmark decomposes the original one-step generative task into three underlying reasoning steps: *change type recognition*, *change localisation*, and *solution identification*. These components represent essential cognitive processes required for understanding intents in code review comments, which is important before generating refined code. By reflecting explicit intermediate reasoning steps, our benchmark provides fine-grained feedback to support model development.

To mitigate potential data contamination, we formulate each reasoning step as a multiple-choice question answering (MCQA) probe. This approach transforms the original task into unfamiliar formats with new solutions, demanding proficiency in code review comprehension rather than sequence memorization (Zhu et al., 2024) and mitigating training data contamination. Furthermore, we leverage MCQA's flexibility to introduce distractor variation strategies, enabling assessment of model understanding across different difficulty levels.

To avoid the noisy data issues present in past benchmarks (Tufano et al., 2024) and ensure high-quality evaluation data, we manually curate 900 valid code refinement examples that cannot be automated by traditional software engineering tools. These examples are sourced from 199 repositories, reflecting nine of the most popular programming languages on GitHub. Finally, we evaluate a wide range of state-of-the-art code-intelligent language models, providing an extensive benchmark to facilitate future research.

2 Background and Related Work

Recently, large language models (LLMs) have shown promise in various software engineering tasks involving natural language artifacts. However, these artifacts vary significantly in their linguistic nature and structure. Some tasks involve explicit, non-conversational language, such as bug reports (Saha et al., 2018) and GitHub issues (Jimenez et al., 2024), which typically contain detailed specifications of defects or feature requests. Other tasks involve static monologues, like commit messages (Jiang et al., 2017), code comments (Hu et al., 2018), and pull request descriptions (Liu et al., 2019), which aim to clearly explain code or code changes.

In contrast, code review is unique as it represents routine conversations in highly collaborative scenarios. As such, they are informal, free-flowing, and can lean on the interlocutor's shared technical knowledge, without being overly specific (Yang et al., 2023). Thus, automating code reviews requires a deep understanding of conversational language in a highly technical context, posing challenges for code review automation. Especially for code refinement where the code submission needs to be revised according to code review comments, which require understanding both the technical implications and the reviewer's unstated expectations. Such nuanced communication makes code review refinement an ideal testbed for evaluating LLMs' ability to bridge technical and conversational understanding in software development.

Code refinement was typically framed as a sequence-to-sequence translation problem, where models "translate" the H_{pre} into a H_{post} based on a review comment R_{nl}. Formally, this problem requires the following estimation:

$$P(H_{post}|H_{pre}, R_{nl}), \quad (1)$$

where H_{pre} denotes the submitted pre-review code hunk, R_{nl} denotes the natural language code review comment, and H_{post} denotes the expected post-review revision of that code hunk. The "hunk" refers to the code snippet within the file, where the code review comment was inlined. See Figure 1 for a concrete example.

While prior work has applied various neural models, such as recurrent neural networks (Tufano et al., 2019), transformers (Tufano et al., 2022; Thongtanunam et al., 2022), the task remains a challenging problem even for recent LLMs such as GPT-4 (Guo et al., 2024; Lu et al., 2023; Tufano et al., 2024).

Indeed, prior work has highlighted several limitations in the evaluation (Guo et al., 2024). Traditional evaluation approaches have relied heavily on text matching metrics such as exact match and BLEU (Tufano et al., 2024; Guo et al., 2024), which are either too strict or fail to provide meaningful feedback. The emergence of LLMs has introduced additional challenges, as they are trained

Code Review Benchmark	Size	#Lang	Metric	DC	MV	VD
Tufano 2021 (Tufano et al., 2021)	1.7k	1	Text Match	✗	✗	✗
T5CR (Tufano et al., 2022)	16.8k	1	Text Match	✗	✗	✗
CodeReviewer (Li et al., 2022)	13.1k	9	Text Match	✗	✗	✗
CodeReview-New (Guo et al., 2024)	14.6k	16	Text Match	✗	✗	✗
CodeReviewQA (Ours)	900	9	Text Match & Probe	✓	✓	✓

DC: Addresses Data Contamination, **MV:** Manual Verification, **VD:** Varied Difficulty

Table 1: Benchmarks for automated code refinement.

on extensive code repositories, creating significant risks of training data contamination in evaluation sets. While some researchers have attempted to address this by collecting code reviews that outpace training cutoff dates (Guo et al., 2024), such approaches lack long-term sustainability. Furthermore, existing benchmarks have been constructed automatically through large-scale mining, and a significant proportion of noise has been reported (Tufano et al., 2024; Liu et al., 2025), undermining the reliability of past results. As a result, all past benchmarks are unsuitable for evaluating the latest models.

Table 1 summarises the limitations in the existing evaluation benchmarks for code refinement, underscoring the need for a new evaluation approach and dataset to reliably assess the capabilities of modern language models. Our proposed **CodeReviewQA** focuses on addressing this gap.

3 CodeReviewQA: Code Review Comprehension Probes

Effective code refinement relies heavily on the ability to comprehend R_{nl} under the context of H_{pre}. Rather than focusing this task as a sequence-to-sequence translation problem like the prior works, we argue that the model must be able to: 1) reason about the type of change R_{nl} is requesting and 2) identify the relevant lines of code in H_{pre} that is the subject of the change; and 3) formulate the required code changes from a wide action space of potential code edits that can be performed on H_{pre}, before generating the refined code H_{post}. The inability to perform the final code generation step may be caused by any failure point amongst this multi-step reasoning process. Additionally, any failure within the intermediary reasoning steps might be propagated from a failure in a prior reasoning step, which obfuscates the specific incompetencies of the model.

To assess the proficiency of a language model in automated code refinement, we design three MCQA probes that replicate the key three intermediate reasoning steps. Below, we describe the construction approach of MCQA of each reasoning step.

Pre-Review Code Submission (H_{pre}):

```
from hypothesistooling . projects . hypothesispython import PYTHON_SRC
from hypothesistooling . scripts import pip_tool , tool_path

PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]

def test_mypy_passes_on_hypothesis() :
```

Code Review (R_{nl}): I think I'd prefer to write these out as literals, unless we can pull them out of the autoupdated CI config? Just thinking about how they'll stay up to date. I think we can also test against 3.11?

What type of change is the code review asking for?
A. Only add new lines of code
B. Only delete existing lines of code
C. Modify the code ✓

Which line numbers is the code review asking to modify code?
A. line number 1 B. line number 2
C. line number 4 ✓ D. line number 6

Which code revision is the code review asking for?
A.
```
4 - PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS >= ["3.7", "3.8", "3.9", "3.10", "3.11"]
```
B.
```
4 - PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS <= ["3.7", "3.8", "3.9", "3.10", "3.11"]
```
C.
```
4 - PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS != ["3.7", "3.8", "3.9", "3.10", "3.11"]
```
D. ✓
```
4 - PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS = ["3.7", "3.8", "3.9", "3.10", "3.11"]
```

Post-Review Code Revision (H_{post}):

```
from hypothesistooling . projects . hypothesispython import PYTHON_SRC
from hypothesistooling . scripts import pip_tool , tool_path

PYTHON_VERSIONS = ["3.7", "3.8", "3.9", "3.10", "3.11"]

def test_mypy_passes_on_hypothesis() :
```

Figure 1: The automated code refinement task with intermediate reasoning steps presented as MCQA probes.

3.1 Change Type Recognition (CTR)

This is a closed set intent classification task that probes the model's ability to infer the intended type of code change. Specifically, given H_{pre}, the model must infer which general type of code change is being requested by R_{nl}. Formally, this problem requires the following estimation:

$$P(C_{type^+}|H_{pre}, R_{nl}) \quad (2)$$

where $C_{type^+} \in \{add, delete, modify\}$ denotes the correct code change type. There are three general types. Firstly, *add* requests involve only adding new lines of code. Secondly, *delete* requests involve only deleting existing lines of code. Lastly, *modify* requests involve altering the existing code by both deleting existing segments and adding new ones. The C_{type^-} distractors are the remaining two incorrect code change types.

This preliminary understanding serves as crucial conditional information that refines the problem

space, providing the correct C_{type} context to subsequently locate where the code changes need to occur and identify what needs to be implemented.

3.2 Change Localisation (CL)

This is a coreference resolution task that probes the model's ability to locate where the intended code change is to occur. Specifically, given R_{nl}, the model must locate the precise lines of code within H_{pre} where the intended C_{type} code change should be applied. Formally, this problem requires the following estimation:

$$P(C_{loc^+}|H_{pre}, R_{nl}, C_{type}) \quad (3)$$

where C_{loc^+} denotes the exact set of line numbers that is the target of the intended code change. When $C_{type} \in \{delete, modify\}$, these are the exact lines of code that need to be deleted or modified. When $C_{type} = \{add\}$, these are the lines of code above where the new code needs to be added. The C_{loc^-} distractors are different sets of lines sampled from H_{pre}. We ensure $|C_{loc^-}| = |C_{loc^+}|$, such that set sizes do not reveal additional information.

As shown in Figure 1, natural code review comments often do not directly specify the exact location of the intended code change, rather this is implicitly conveyed based on a shared understanding between the reviewer and code author. Thus, the model must possess the ability to conduct anaphora resolution across modalities, between anaphors in R_{nl} and antecedents in H_{pre}. Inferring the incorrect C_{loc}, would subsequently hinder the model's ability to identify the H_{post} that accurately reflects the intended code change.

3.3 Solution Identification (SI)

This task probes the model's ability to both conduct open intent extraction from R_{nl} and identify the H_{post} that accurately reflects that intent. Given R_{nl}, the model must identify the correct H_{post} that reflects the intended C_{type} change on C_{loc} in H_{pre}. The intuition behind this task design is that if a model is able to generate a correct H_{post+} revision, it should at least be able to identify that exact H_{post+} solution amongst a solution space with incorrect H_{post-} alternatives. Formally, this problem requires the following estimation:

$$P(H_{post+}|H_{pre}, R_{nl}, C_{type}, C_{loc}), \quad (4)$$

where H_{post+} denotes the diff of the ground truth post-review code revision. We only include cases where $C_{type} \in \{add, modify\}$, as $\{delete\}$ cases merely delete C_{loc} located in the previous task.

3.4 Variation of Distractor Difficulty

The MCQA format allows flexibility in varying the difficulties of the distractors (i.e., the incorrect answer options). This not only allows us to stress test the models' level of understanding, but also enables the ability to evolve the benchmark against performance saturation. We specify the process of generating easy and hard distractors for *Change Localisation* and *Solution Identification*, as these tasks allow for variation in solutions.

Change Localisation Distractors. We vary the difficulty based on the degree of overlap between the sets of the provided C_{loc} options. For the easy distractors, we sample C_{loc^-} distractors from H_{pre}, such that the *Jaccard Similarity* between all answer options are as low as possible. This ensures that all answers are easy to distinguish from each other and the ground truth is more obvious to locate. For the hard distractors, we sample C_{loc^-} distractors, such that the *Jaccard Similarity* between all answer options are as high as possible. This ensures that all answers are hard to discern from the ground truth, and requires the model to locate every exact line of the intended code change.

Solution Identification Distractors. To create distractor options H_{post-}, we generate modified versions of H_{post+} by perturbing code elements in the change location C_{loc}, ensuring the intended code change is no longer correctly implemented. To create plausible but incorrect distractors that imitate possible mistakes that the models would make, we use a surrogate language model[2] to 1) identify the code element with the highest average token surprisal in the correct H_{post+} solution, 2) mask it, and then retroactively fill the masks with diverse candidates. We keep candidates that are not equivalent to H_{post+} as valid H_{post-} distractor candidates. All generated distractors are manually verified for semantic in-equivalence to the ground truth. The algorithm of constructing H_{post-} distractors is illustrated in Algorithm 1 in Appendix.

We vary the difficulty based on the degree of semantic similarity between the H_{post^-} distractors and the H_{post^+} ground truth. For the easy distractors, we retain the H_{post^-} distractors which yield the lowest *cosine similarity* against H_{post^+} in the

[2]We use a competitive surrogate model that is proficient in coding (Codestral-22B-v0.1) with a temperature of 3.5

embedding space of the surrogate model. This ensures that each H_{post^-} is substantially different to H_{post^+}, such that it is easy to discern. For the hard distractors, we retain the H_{post^-} distractors which yield the highest *cosine similarity* against H_{post^+}. This ensures that each H_{post^-} is only marginally different from H_{post^+}, such that it is hard to discern. See Figure 3 in the Appendix for examples of variation in difficulty.

4 Dataset Preparation

Data Source. We built our benchmark based on the most recently published automated code refinement dataset (Guo et al., 2024). This multilingual dataset was constructed from code reviews that occurred after January 1, 2022. To ensure that we have a sizable amount of clean data for each of the programming languages in our benchmark, we only include the nine most popular programming languages on GitHub i.e. C, C++, C#, Go, Java, Javascript, PHP, Python, Ruby. These 9,367 examples were mined from 259 repositories, filtered from a list of the most starred GitHub projects.

Data Sampling. To ensure diversity and quality in our benchmark, we conducted stratified sampling (Baltes and Ralph, 2022) across all nine programming languages in the dataset, and discarded any examples that were noisy or unfaithfully represented the task of code refinement. For each of the nine languages, we sampled until there were 100 clean examples each, resulting in 900 total examples in our benchmark. Within each language partition, we also conducted stratified sampling across projects to maintain diversity. This mitigated bias towards the code reviews of any specific project, the nature of which are influenced by their particular software development tools (Paschali et al., 2017), processes (Viggiato et al., 2019) and issues (Linares-Vásquez et al., 2014).

Data Curation. We discard examples that were noisy or unfaithfully represented the task of code refinement. The noisy examples refer to code review comments that are unclear, ignored, no change asked, or linking to wrong code hunks. See Appendix A for a detailed explanation of these noise types. These kind of review comments were reported as critical quality issues with existing code review datasets by prior work (Tufano et al., 2024). Unfaithful examples refer to the scenarios that do not faithfully represent the automated code refinement task i.e., reviews directly including the entire *intended code revision implementation*, reviews regarding *code formatting*, reviews that are *not self-contained* (Tufano et al., 2024; Lin et al., 2024). Instead, examples in the benchmark should represent meaningful quality improving code reviews that are beyond the capacity of traditional rule-based software engineering tools. See Appendix B for a detailed explanation of these unfaithful examples.

To discard noisy and unfaithful examples, we first applied heuristic filters as detailed in Appendix C, before manual verficiation. This resulted in 3,761 out of 9,367 examples being discarded from the source dataset. The manual discarding was conducted by two annotators who are currently pursuing a PhD in software engineering. Both annotators independently annotated 3k examples, and resolved all conflicts together across 46 rounds. The μ and σ of the Cohen's Kappa were 0.89 and 0.11, respectively. For C, JavaScript and Ruby, less than 100 clean examples could be obtained from the source dataset, thus, the remaining examples were sampled from code reviews conducted in 2021 (Li et al., 2022). The overall retention rate was 13%, highlighting the critical quality issue in the original dataset, necessitating curating automated code refinement benchmarks for accurate and reliable evaluation. The final benchmark includes 199 of the original 259 GitHub repositories. Table 4 in the Appendix shows benchmark statistics.

5 Experimental Setup

5.1 MCQA Setup

To support the MCQA probes for our CodeReviewQA, we detail our prompt design, answer extraction approach, and evaluation framework incorporating invariance testing.

Prompt. We use multiple-choice prompting that takes an input containing three components: task definition, question, and options. The task definition specifies the broad purpose (e.g., "tests code review comprehension"). The question section presents the code review scenario within a structured template that includes programming language markers, the preview code H_{pre}, and the code review comment R_{nl}. Finally, the options section lists multiple choice answers labeled alphabetically (A, B, C, D), with explicit instructions to respond with only the letter symbol. This prompt structure is used across all three tasks while varying only the specific task parameters and number of answer op-

tions. See Figure 2 in the Appendix for all prompt templates used.

Answer Extraction. We use multiple choice prompting with a max output length of one, where the symbol token $\in \{A, B, C, D\}$ with the highest log probability is considered as the selected answer. This style of prompting avoids the conflation of likelihood of sequence and likelihood of answer, eliminates the need for normalisation and allows for direct comparison between answers (Robinson and Wingate, 2023). Our implementation uses the vLLM inference framework (Kwon et al., 2023) with guided decoding targeting the option symbols.

Invariant Test and Evaluation. To reduce the likelihood of random correct guesses, for each question, we exposed the models to every order combination of the answer options. This resulted in $N!$ runs per question, where N is the number of answer options provided. Thus, the likelihood of guessing the correct answer for all combinations of a question is merely $(\frac{1}{N})^{N!}$. With this, we mitigate the models' invariability in selecting the correct answer, regardless of the position of that answer. To be counted as correctly answering that question, the models must select the correct answer for all $N!$ runs, which is a more reliable indicator of the models' understanding (Wang et al., 2024).

5.2 Model Selection Criteria

We list the criteria that determines whether a LLM is appropriate for this benchmark.

MCQA Proficiency. The LLM must have achieved state-of-the-art results in MCQA style benchmarks e.g., MMLU (Hendrycks et al., 2021). This accounts for format as a confounding factor.

MCSB Proficiency. The LLM must demonstrate proficiency in multiple choice symbol binding (MCSB; Robinson and Wingate (2023)). This ensures that the answer extraction method is not a confounding factor. We report the Proportion of Plurarity Agreement (PPA), which measures the degree of order invariance in selecting the symbol of the plurarity answer. Formally, PPA is calculated as the average of $\frac{k}{N!}$ over a dataset, where k is the number of times the plurarity answer's symbol yielded the highest log probability for a given question and $N!$ is the aforementioned number of order combinations for N answer options. MCSB proficiency is demonstrated when a PPA significantly higher than the random baseline of $\frac{1}{N}$ is achieved.

Coding Proficiency. In addition to understanding the natural language in R_{nl}, the model must also be able to understand the code in H_{pre} and H_{post}. Therefore, the LLM must have demonstrated proficiency in coding related benchmarks e.g., HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021).

In total, we select 65 state-of-the-art open source LLMs, that have satisfied the three criteria. The included models are considered state-of-the-art as of January, 2025. See Table 6 in Appendix F for descriptions of all 65 models. The models are grouped into five scales based on their model parameters: $\leq$3B, $\leq$9B, $\leq$16B, $\leq$34B, and $\leq$72B. We select models under 72B as it is the largest size we can run locally to extract answer probabilities.

6 Results

To compare model capacity differences in the automated code refinement (ACR) tasks and three MCQA probe tasks, we conducted experiments using all 65 selected models. Due to space limitations, detailed results are provided in Appendix F. We summarize the key observations of top-2 models of each scale class in Table 2.

ACR vs. Probes.[3] In general, we find that larger language models tend to achieve higher exact match rates on average in automated code refinement. However, their performance on probing tasks could vary. Table 2 (column ACR) shows that Llama-3.1-70B-Instruct achieved the highest exact match rate of 50.3%. Interestingly, Qwen2.5-72B-Instruct achieves an exact match rate 2 percentage points lower, but outperforms Llama-3.1-70B-Instruct in change type recognition (CTR) and solution identification (SI). Our benchmark reveals similar results for the smaller models. For example, Qwen2.5-Coder-14B-Instruct, gemma-2-27b-it, and QwQ-32B-Preview achieve comparable exact match rates in ACR with less than a percentage point difference. However, their performance on each probing task is substantially different. This highlights the benefits of a more granular assessment of model capabilities, beyond exact match rates, as different models exhibit varying strengths across specific probing tasks.

Below, we discuss model capabilities in these three probing tasks.

CTR Results.[4] Interestingly, we find that most of the $\leq$ 3B models were already competent in this

[3]Table 7 in Appendix F presents the full list of automated code refinement results.

[4]Table 8 in Appendix F shows the full results for change type recognition.

Model	ACR	CTR	CL_E	CL_H	SI_E	SI_H
Llama-3.2-3B-Instruct	25.9	78.8	0.8	0.3	9.9	7.6
Qwen2.5-Coder-3B-Instruct	30.3	77.7	1.8	1.6	12.2	8.0
Qwen2.5-Coder-7B-Instruct	41.0	78.6	13.8	10.7	67.6	55.2
gemma-2-9b-it	39.0	74.1	59.2	52.0	58.8	49.6
CodeLlama-13b-Instruct-hf	36.7	67.8	0.11	0.1	13.8	10
Qwen2.5-Coder-14B-Instruct	46.6	73.9	46.7	37.3	65.5	56.2
gemma-2-27b-it	46.4	74.0	70.1	58.7	76.2	65.7
QwQ-32B-Preview	45.6	60.3	52.1	50.1	79.1	75.1
Llama-3.1-70B-Instruct	**50.3**	68.4	**74.7**	**69.0**	84.2	76.7
Qwen2.5-72B-Instruct	48.7	**79.8**	64.2	58.3	**97.1**	**90.9**

ACR: Automated Code Refinement, **CTR:** Change Type Recognition
CL: Change Localisation, **SI:** Solution Identification, **E:** Easy, **H:** Hard

Table 2: Top-2 Performing Models (per scale class) based on Exact Match (%) in Automated Code Refinement.

task, with Llama-3.2-3B-Instruct achieving 78.8% invariant accuracy. Despite the promising results of small models, this ability plateaus as we scale model size. In fact, the best performance from the $\leq$ 72B models was only 79.8% by Qwen2.5-72B-Instruct, which is only a 1% increase.

CL Results.[5] Overall, change localisation tend to be the most difficult reasoning task in the benchmark. Most of the $\leq$ 3B models achieved invariant accuracies of between 0%-3% for both variations. The only exceptions being Qwen2.5-3B-Instruct and Phi-3-mini-128k-instruct, which could achieve 39.3% and 34.1% for the easy variation, respectively. In contrast, we find that many models from the $\leq$ 34B and $\leq$ 72B classes could achieve invariant accuracies of more than 70% for the easy variation and more than 60% for the hard variation.

SI Results.[6] Similarly, we also find that competency in this task strengthens with size, with the exception of a few anomalies that far outperform their scale class average. For example, Phi-3-mini-128k-instruct can achieve an invariant accuracy of 58.16% in the easy variation, whilst the majority of $\leq$ 3B models achieve less than 13%. In contrast, many models from the $\leq$ 72B class can achieve more than 80% in the easy variation and more than 70% for hard. Most notably, Qwen2.5-72B-Instruct, could achieve a near perfect score of 97.06% for the easy variation, and 90.88% for the hard variation, despite previously achieving underwhelming results for change localisation.

[5]Tables 9 and 10 in Appendix F present the full results for the easy and hard variations of change localisation.

[6]Tables 11 and 12 in Appendix F show the full results for the easy and hard variations of solution identification.

7 Evaluating Data Contamination

To what extent is CodeReviewQA resistant to data contamination? We utilise two canonical metrics for measuring data contamination, perplexity (Jelinek et al., 1977) and n-gram accuracy (Xu et al., 2024). Perplexity is an information-theoretic metric, which quantifies the uncertainty of a language model in a token sequence (Jelinek et al., 1977), which can be formulated as:

$$\text{PPL}(\mathbf{X}) = \exp(-\frac{1}{t}\sum_{t=0}^{t}\log p_\theta(x_i|x_{<i})), \quad (5)$$

where $\mathbf{X} = [x_0, x_1, ..., x_t]$ denotes a tokenised sequence. In our case, the sequence is a concatenation of the question, i.e., prompt including H_{pre} and R_{nl} and the solution, i.e. H_{post} for automated code refinement or answer options for MCQA probes. A low perplexity score indicates high confidence, whilst a high perplexity score indicates low confidence. Unusually low perplexity scores may indicate data contamination. N-gram accuracy measures the model's ability to predict random n-gram sequences from K starting points that are uniformly sampled from an example (Xu et al., 2024), i.e., the aforementioned sequence X. It is calculated by the following equation:

$$\text{NG}(\mathbf{X}) = \frac{1}{\eta \cdot K}\sum_{i=0}^{\eta}\sum_{j=0}^{K} I(X_{s_j:s_j+n}, \hat{X}_{s_j:s_j+n}), \quad (6)$$

where η denotes the corpus size, i denotes the i_{th} sequence in the corpus, s_j denotes the index of the j^{th} starting point, $X_{s_j:s_j+n}$ denotes the ground truth n-gram to be predicted and I denotes an indicator function that applies exact match. Unusually, a high n-gram accuracy may indicate data contamination. Following prior work (Xu et al., 2024), we set K = 5 and $n = 5$ to measure 5-gram accuracy.

For this experiment, we use the largest and newest models that we can support from the most popular model families, as they are most likely to exhibit memorisation (Kiyomaru et al., 2024). We use base versions of models, as instruction-tuned versions are optimised for responding to prompts rather than completing sequences verbatim.

To test the effectiveness of MCQA reformulation in mitigating data contamination, we compare our benchmark in MCQA probe form with the original automated code refinement form, as well as

Benchmark	Task	Llama-3.1-70B		Qwen2.5-72B	
		PPL	NG_5	PPL	NG_5
CodeReviewer	ACR	4.1	28.1	3.6	30.7
CodeReview-New	ACR	4.4	40.3	3.9	42.6
CodeReviewQA	ACR	4.5	40.3	4.1	42.0
	MCQA	**6.0**	**25.1**	**5.4**	**26.8**

ACR: Automated Code Refinement, **MCQA**: Multiple Choice Question & Answer

Table 3: Comparing Perplexity Scores (PPL) and 5-gram Accuracies (NG_5 %) on our CodeReviewQA against existing code review benchmarks.

the most widely used automated code refinement benchmarks, i.e. CodeReviewer (Li et al., 2022) and CodeReview-New (Guo et al., 2024).

Table 3 shows the perplexity and 5-gram accuracy on the three benchmarks, based on two popular base models Llama-3.1-70B and Qwen2.5-72B. We find that perplexity on the older CodeReviewer benchmark is far lower than on CodeReview-New, yet the 5-gram accuracies are also lower. A likely explanation is that older code reviews have been extensively included in the models' pre-training and resembles the vast majority of the corpus, however, since they may not have been included in the latter stages of training, there is less verbatim memorisation of the examples (Kiyomaru et al., 2024). In contrast, the newer code reviews may represent a distribution shift, yet is more likely to be included in the latter stages of training, thus concurrently inducing higher perplexity and higher verbatim memorisation. We find that **CodeReviewQA** in the original automated code refinement format yields similar results to CodeReview-New, which is within expectation as one is simply a curated subset of the other. However, when reformulating into the MCQA probe format, our benchmark yields significantly higher perplexity than all past benchmarks with lower 5-gram accuracies, despite using the same examples. Therefore, we find that MCQA reformulation with synthetic questions and answers does mitigate the effects of data contamination, allowing for the reuse of code reviews that may have been previously included in pre-training. Coinciding with our insights from probing, models that perform well on automated code refinement with only memorisation, may be exposed when evaluated with MCQA probes of the same examples.

8 Conclusion

In this study, we focus on evaluating recent large language models' capabilities through automated code refinement, a challenging task in code review. We addressed two key limitations in existing work: the inability of exact match metrics to provide fine-grained performance feedback and the potential for data contamination. To this end, we propose CodeReviewQA as a new benchmark, which consists of 900 high-quality manually curated code review samples. We reformulated the code refinement generation task into three intermediate reasoning tasks and designed multiple-choice questions to probe model performance. Our experimental results across 72 large language models revealed capability differences that traditional evaluation metrics failed to capture. Additionally, our data contamination evaluation demonstrated that the CodeReviewQA dataset effectively mitigates training data contamination concerns.

9 Limitations

The size of dataset. Our CodeReviewQA has a relatively modest size resulting from high noise in existing benchmarks and rigorous manual verification. Despite its size, our CodeReviewQA is diverse and comprehensive, covering real-world code reviews in nine programming languages and from 199 projects.

The construction of distractors. The change localisation task focused on the line level rather than more fine-grained level (e.g., token level). However, our findings show that many models struggle with identifying the location of changes even at the line level. Future work can further explore approaches to automatically construct and evaluate localisation at the token level.

The interaction among tasks. We did not investigate the causal relationships between these tasks, meaning that failure in one probe task does not necessarily predict performance on another. However, our experimental results demonstrate that analyzing performance across all three probe tasks alongside the automated code refinement task provides more comprehensive insights into model weaknesses.

The diversity of prompts. We used the same prompt and hyperparameters for each task to maintain consistent, comparable results across models. Different prompts might impact model performance. However, our main focus was not to find the optimal prompt for each individual model, but to investigate systematic differences across models on comprehension and generation tasks.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: A critical review and guidelines. *EMSE*, 27(4):94.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Vasiliki Efstathiou and Diomidis Spinellis. 2018. Code review comments: language matters. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '18, page 69–72, New York, NY, USA. Association for Computing Machinery.

Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the potential of chatgpt in automated code refinement: An empirical study. In *ICSE*, pages 1–13, New York, NY, USA. IEEE.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *ICLR*. OpenReview.net.

Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*, pages 200–210. ACM.

Frederick Jelinek, Robert L. Mercer, Lalit R. Bahl, and Janet M. Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *Journal of the Acoustical Society of America*, 62.

Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 135–146. IEEE Computer Society.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Hirokazu Kiyomaru, Issa Sugiura, Daisuke Kawahara, and Sadao Kurohashi. 2024. A comprehensive analysis of memorization in large language models. In *Proceedings of the 17th International Natural Language Generation Conference*, pages 584–596, Tokyo, Japan. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

VI Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Proceedings of the Soviet physics doklady*.

Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, and Neel Sundaresan. 2022. Automating code review activities by large-scale pre-training. In *ESEC/FSE*, page 1035–1047, New York, NY, USA. ACM.

Hong Yi Lin, Patanamon Thongtanunam, Christoph Treude, and Wachiraphan Charoenwet. 2024. Improving automated code reviews: Learning from experience. In *MSR*, pages 278–283, New York, NY, USA. IEEE.

Mario Linares-Vásquez, Sam Klock, Collin McMillan, Aminata Sabané, Denys Poshyvanyk, and Yann-Gaël Guéhéneuc. 2014. Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps. In *ICPC*, page 232–243, New York, NY, USA. ACM.

Chunhua Liu, Hong Lin, and Patanamon Thongtanunam. 2025. Too noisy to learn: Enhancing data quality for code review comment generation. In *22nd International Conference on Mining Software Repositories*.

Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 176–188. IEEE.

Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In *34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023, Florence, Italy, October 9-12, 2023*, pages 647–658. IEEE.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, Philadelphia, PA, USA. ACL.

Maria-Eleni Paschali, Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, and Ioannis Stamelos. 2017. Reusability of open source software across domains: A case study. *J. Softw.*, 134:211–227.

Chanathip Pornprasit and Chakkrit Tantithamthavorn. 2024. Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology*, 175:107523.

Mohammad Masudur Rahman, Chanchal K. Roy, and Raula Gaikovina Kula. 2017. Predicting usefulness of code review comments using textual features and developer experience. In *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 215–226. IEEE Computer Society.

Joshua Robinson and David Wingate. 2023. Leveraging large language models for multiple choice question answering. In *ICLR*. OpenReview.net.

Ripon K. Saha, Yingjun Lyu, Wing Lam, Hiroaki Yoshida, and Mukul R. Prasad. 2018. Bugs.jar: a large-scale, diverse dataset of real-world java bugs. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 10–13, New York, NY, USA. Association for Computing Machinery.

June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the silence: the threats of using llms in software engineering. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, NIER@ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 102–106. ACM.

Patanamon Thongtanunam, Chanathip Pornprasit, and Chakkrit Tantithamthavorn. 2022. Autotransform: automated code transformation to support modern code review process. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 237–248, New York, NY, USA. Association for Computing Machinery.

Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On learning meaningful code changes via neural machine translation. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 25–36. IEEE / ACM.

Rosalia Tufano, Ozren Dabić, Antonio Mastropaolo, Matteo Ciniselli, and Gabriele Bavota. 2024. Code review automation: Strengths and weaknesses of the state of the art. *TSE*, 50(2):338–353.

Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 2291–2302. ACM.

Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards automating code review activities. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 163–174. IEEE.

Markos Viggiato, Johnatan Oliveira, Eduardo Figueiredo, Pooyan Jamshidi, and Christian Kästner. 2019. Understanding similarities and differences in software development practices across domains. In *ICGSE*, pages 84–94, New York, NY, USA. IEEE.

Haochun Wang, Sendong Zhao, Zewen Qiang, Bing Qin, and Ting Liu. 2024. Beyond the answers: Reviewing the rationality of multiple choice question answering for the evaluation of large language models. *CoRR*, abs/2402.01349.

Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. 2024. Benchmarking benchmark leakage in large language models. *CoRR*, abs/2404.18824.

Lanxin Yang, Jinwei Xu, Yifan Zhang, He Zhang, and Alberto Bacchelli. 2023. Evacrc: Evaluating code review comments. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 275–287, New York, NY, USA. Association for Computing Machinery.

Wenhong Zhu, Hongkun Hao, Zhiwei He, Yunze Song, Jiao Yueyang, Yumeng Zhang, Hanxu Hu, Yiran Wei, Rui Wang, and Hongyuan Lu. 2024. CLEAN-EVAL: clean evaluation on contaminated large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 835–847. Association for Computational Linguistics.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du,

Harm de Vries, and Leandro von Werra. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *CoRR*, abs/2406.15877.

A Data Quality Issue Details

We explain the four types of data quality issues found in code review datasets. Firstly, *unclear comments* are review comments where even humans cannot comprehend the intended change. Secondly, *no change asked* refers to review comments that are not actionable. Thirdly, *ignored comment* are examples where the developer ignores the review comment, resulting in a post-review code revision H_{post} that does not reflect the intended code change. Lastly, *wrong linking* refers to data mining issues, where the review comment is not related to the paired pre-review code submission H_{pre}.

B Unfaithful Example Details

We explain the three types of unfaithful examples found in code review datasets. Firstly, some code reviews directly include the entire *intended code revision implementation*. These cases can be resolved by directly copy and pasting from the review itself, which does not assess natural language comprehension. Secondly, *code formatting* related examples can already be resolved by linters and therefore are not useful to learn. These examples also fail to assess the models' ability in handling challenging and meaningful code reviews. Thirdly, code reviews that are *not self-contained* require information beyond the provided code hunk H_{pre} to understand, thus, it is impossible for the model (or even a human) to intuit the intended code change.

C Heuristic Filtering Details

We conducted keyword-based filtering to automatically discard examples that clearly violate the data quality and faithfulness issues mentioned above. With regards to *unclear comments*, we discarded reviews with less than 10 characters, since they are likely to be too short to convey a code change requirement. We also removed reviews that did not contain any alphabetic characters. With regards to code reviews that already contain the *intended code revision implementation*, we discarded reviews that included the "“‘" GitHub code block indicator. With regards to code reviews that are only demanding *code formatting* changes, we removed reviews that mention "indentation", "spacing" and "lint". With regards to reviews that are *not self-contained*, we removed reviews that mention "revert", "as above" and "ditto". We manually inspected all discarded examples to ensure that no false positives were detected by our filter.

D Descriptive Statistics

We explain the descriptive statistics used in this study for describing the benchmark.

Comment length is measured by the number of whitespace separated words in the code review comment. Longer comments may contain more complex requirements, explanations or other discussions. Table 4 shows that the average comment length is 18 words. See Figure 4 for examples of different comment lengths.

Code edit distance represents the size of the code change between H_{pre} and H_{post}. Given that some examples only involve changes in the inlined code comment, we use the more general Levenshtein distance (Levenshtein, 1966) to measure the number of character edits between the two versions of code. Table 4 shows that the average code edit distance is 56 characters. See Figure 5 for examples of different code edit distances.

Change locations is the number of lines involved in the change, as discussed in the task of change localisation. Table 4 shows the change location statistics. For the 35 (4% of total) code reviews that request to *add* code, the average number of change locations is 1 line. For the 144 (16% of total) code reviews that request to *delete* code, the average number of change locations is 3 lines. For the 721 (80% of total) code reviews that request to *modify* code, the average number of change locations is 2 lines.

Code element ratio is the proportion of tokens in the code review comment that are code elements. It is calculated as $\frac{\text{Code elements}}{\text{Comment length}}$. Reviewers may use code elements in conjunction with natural language to describe the intended code change. Comments with a higher proportion of code tokens may be more explicit in their specification of the requirements (Rahman et al., 2017). Table 4 shows that the average code element ratio is 0.09. See Figure 6 for examples of different code element ratios.

Specification ratio is the code edit distance of the change divided by the length of its respective code review comment. It is calculated as $\frac{\text{Code edit distance}}{\text{Comment length}}$, and can be interpreted as the number of character edits in the code with respect to each word in the comment. Since code review com-

Statistic	Min	Max	Mean	SD	Q1	Median	Q3
Comment Length	2	98	18	15	8	13	23
Code Edit Distance	2	827	56	85	9	25	67
Change Locations							
Add (35)	1	2	1	0	1	1	1
Delete (144)	1	19	3	4	1	2	4
Modify (721)	1	15	2	2	1	1	2
Code Element Ratio	0.00	0.89	0.09	0.15	0.00	0.00	0.13
Specification Ratio	0.04	165.40	4.88	9.79	0.67	1.83	5.07

See Appendix D for a detailed explanation of the descriptive statistics.

Table 4: **CodeReviewQA** descriptive statistics.

Model	%Fail ≥ 1 Probe	CTR	CL	SI
Qwen2.5-Coder-3B-Instruct	**99.8**	23.4	**99.4**	**95.8**
Qwen2.5-Coder-7B-Instruct	96.6	22.2	93.0	55.0
Qwen2.5-Coder-14B-Instruct	87.3	29.3	74.0	56.0
gemma-2-27b-it	75.5	25.7	53.7	41.4
Llama-3.1-70B-Instruct	76.5	**37.4**	46.3	32.7

CTR: Change Type Recognition, **CL:** Change Localisation, ≥ 1 **Probe:** Failed at least one probe, **SI:** Solution Identification

Table 5: MCQA Failure (%) in Non-Exact Match Cases on top-performing models at each scale.

ments may be under-specified and implicit, we use specification ratio as a heuristic metric that incorporates this notion. Intuitively, examples with large code edit distances and short comment lengths i.e. large specification ratio, may be under-specified in its description of the required code change. Table 4 shows that the average specification ratio is 4.88. See Figure 7 for examples of different specification ratios.

E Insights from Probing

What model weaknesses does CodeReviewQA expose, beyond those identified by automated code refinement? We use our MCQA probes to investigate failure cases of the top-performing models in Table 2 by analyzing examples where model outputs on the code refinement task do not exactly match the ground truth refined code. For each failed case from each model, we examine whether the model succeeds or fails on the three MCQA tasks. Through this analysis, we aim to identify specific weaknesses in models' code review comprehension abilities that may contribute to their failures in automated code refinement.

The results are presented in Table 5, including the percentage of failure on each of the MCQA tasks,[7] and the overall failure on at least one task ($\geq$ 1 Probe). We find that all five models failed at least one MCQA probe for 76%-100% of the examples, indicating a varying level of difficulty in comprehending the code review. Change type recognition is seldom a root cause, as failure in this capability only accounts for 22%-37% of cases. In contrast, change localisation is often a root cause, particularly for the three smaller models (74%-99% failures). When analysing the scenarios that requested to add or modify code, solution identification failures account for 96% of cases for Qwen2.5-Coder-3B-Instruct. Thus, for the smallest model, most non-exact matches can be accounted for by failures in both change localisation and solution identification. For both Qwen2.5-Coder-7B/14B-Instruct, change localisation is the major flaw. For the two largest models, the cause of failure is more evenly distributed, varying on an individual basis.

On the other hand, we also analyze the successful cases where the model achieves an exact match. Intuitively, if a model can achieve exact match on an example, it should be able to fully comprehend the code review, thus achieving a perfect score across the reasoning probes. However, the models often could not accurately answer all probes for 49%-99% of their exact matches. Interestingly, this trend shows a strict inverse relationship with model size, where smaller models yield higher proportions of exact matches that the model could not consistently complete all probes for. These symptoms indicate signs of rote memorisation.

F Experimental Results on 65 models

[7] For change location and solution identification, a failure is counted if a model fails in either easy or hard mode.

Algorithm 1: Create H_{post^-} Distractors for Solution Identification

Input: Surrogate LLM f_θ, Temperature k, No. of Distractors N, Difficulty ϕ
Output: Set of H_{post^-} Distractors D

// Identify code elements in the changed lines of the post-review code revision
$Lines \leftarrow$ GetChangedLines(H_{post^+});
$AST \leftarrow$ GetAbstractSyntaxTree(H_{post^+});
$Nodes \leftarrow$ GetLeafNodes($AST, Lines$);
// Calculate average token surprisal for each identified code element
$S_{token}, S_{node}, C_{distractor}, D \leftarrow \emptyset, \emptyset, \emptyset, \emptyset$;
for $c_t \in H_{post^+}$ **do**
  $\boldsymbol{h}_{t-1} \leftarrow f_\theta(H_{pre}, R_{nl}, c_{<t})$;
  $S_{token}[c_t] \leftarrow -\log_2 P(c_t|\boldsymbol{h}_{t-1})$;
for $n_t \in Nodes$ **do**
  for $c_t \in n_t$ **do**
    $S_{node}[n_t] \leftarrow S_{node}[n_t] + S_{token}[c_t]$;
  $S_{node}[n_t] \leftarrow \frac{S_{node}[n_t]}{length(n_t)}$;
// Mask the code element with the highest average token surprisal
$n_{max} \leftarrow$ GetMaxKeys($S_{node}, 1$);
$Mask \leftarrow$ ApplyMask(H_{post^+}, n_{max})
// Create a set of distractors for each of the easy and hard variations
while $length(C_{distractor}) < 2 \times N$ **do**
  $\hat{n}_{max} \leftarrow \arg\max_{n_{max}} P_k(n_{max}|f_\theta(Mask))$;
  $Candidate =$ InFill($Mask, \hat{n}_{max}$);
  if $Candidate \neq H_{post^+} \wedge Candidate \notin C_{distractor}$ **then**
    $\theta =$ Cosine($f_\theta(Candidate), f_\theta(H_{post^+})$);
    $C_{distractor}[Candidate] \leftarrow \theta$;
// Select distractors most semantically different from ground truth for easy and distractors most semantically similar to ground truth for hard
if $\phi = easy$ **then**
  $D \leftarrow$ GetMinKeys($C_{distractor}, N$);
else if $\phi = hard$ **then**
  $D \leftarrow$ GetMaxKeys($C_{distractor}, N$);

Automated Code Refinement

{lang} = C/CPP/CSharp/Go/Java/JavaScript/PHP/Python/Ruby

The following {lang} code snippet has received a code review.

[{lang}]

{code_snippet}

[/{lang}]

[CODE REVIEW]

{code_review}

[/CODE REVIEW]

Please generate a revised version of the code snippet according to the code review. Do not add explanations.

[{lang}]

Change Type Recognition

{option_a}, {option_b}, {option_c} = only add new lines of code/only delete existing lines of code/modify the code

The following is a multiple choice question (with answers) that tests code review comprehension.
Question: Given this {lang} code snippet, what type of change is the code review asking for?

[{lang}]

{code_snippet}

[/{lang}]

[CODE REVIEW]

{code_review}

[/CODE REVIEW]

Possible answers:
A. {option_a}

B. {option_b}

C. {option_c}

Answer with the letter symbol only. Answer:

Change Localisation

{change_type} = add new lines of code under/delete code/modify code

The following is a multiple choice question (with answers) that tests code review comprehension.
Question: Given this {lang} code snippet, which line numbers is the code review asking to {change_type}?

[{lang}]

{code_snippet}

[/{lang}]

[CODE REVIEW]

{code_review}

[/CODE REVIEW]

Possible answers:
A. {option_a}

B. {option_b}

C. {option_c}

D. {option_d}

Answer with the letter symbol only. Answer:

Solution Identification

The following is a multiple choice question (with answers) that tests code review comprehension.

Question: Given this {lang} code snippet, which code revision is the code review asking for?

[{lang}]

{code_snippet}

[/{lang}]

[CODE REVIEW]

{code_review}

[/CODE REVIEW]

Possible answers:
A. {option_a}

B. {option_b}

C. {option_c}

D. {option_d}

Answer with the letter symbol only. Answer:

Figure 2: Prompt Templates.

Pre-Review Code Submission

```
1       def main(args: argparse.Namespace):
2                   )
3           host_environment = host_environments.pop()
4
5           module_dir_paths = sort_and_dedup_paths([
6               iree_artifacts.get_module_dir_path(config.module_generation_config)
7               for config in run_configs
8           ])
9
10          output_map[device_name] = {
11              "host_environment": dataclasses.asdict(host_environment),
```

Code Review: Huh, would be nice if the path was just naturally serializable

---

Which line numbers is the code review asking to modify code? A. line numbers 5, 6, 8 ✓

Change Localisation (Easy)

B. line numbers 1, 2, 3 C. line numbers 1, 4, 5 D. line numbers 1, 6, 7

Change Localisation (Hard)

B. line numbers 1, 5, 6 C. line numbers 1, 5, 8 D. line numbers 1, 6, 8

Which code revision is the code review asking for?

A. ✓

```
5 -     module_dir_paths = sort_and_dedup_paths([
5 +     module_dir_paths = sorted(set(
6 -         iree_artifacts.get_module_dir_path(config.module_generation_config)
6 +         str(iree_artifacts.get_module_dir_path(config.module_generation_config))
8 -     ])
8 +     ))
```

Solution Identification (Easy)

B.

```
5 -     module_dir_paths = sort_and_dedup_paths([
5 +     module_dir_paths = sorted(set(
6 -         iree_artifacts.get_module_dir_path(config.module_generation_config)
6 +         str(struct_lucule.get_module_dir_path(config.module_generation_config))
8 -     ])
8 +     ))
```

C.

```
5 -     module_dir_paths = sort_and_dedup_paths([
5 +     module_dir_paths = sorted(set(
6 -         iree_artifacts.get_module_dir_path(config.module_generation_config)
6 +         str(assertTrue_localhost.get_module_dir_path(config.module_generation_config))
8 -     ])
8 +     ))
```

D.

```
5 -     module_dir_paths = sort_and_dedup_paths([
5 +     module_dir_paths = sorted(set(
6 -         iree_artifacts.get_module_dir_path(config.module_generation_config)
6 +         str(indexChat_retry.get_module_dir_path(config.module_generation_config))
8 -     ])
8 +     ))
```

Solution Identification (Hard)

B.

```
5 -     module_dir_paths = sort_and_dedup_paths([
5 +     module_dir_paths = sorted(set(
6 -         iree_artifacts.get_module_dir_path(config.module_generation_config)
6 +         str(View_DEF.get_module_dir_path(config.module_generation_config))
8 -     ])
8 +     ))
```

C.

```
5 -     module_dir_paths = sort_and_dedup_paths([
5 +     module_dir_paths = sorted(set(
6 -         iree_artifacts.get_module_dir_path(config.module_generation_config)
6 +         str(develop_weight.get_module_dir_path(config.module_generation_config))
8 -     ])
8 +     ))
```

D.

```
5 -     module_dir_paths = sort_and_dedup_paths([
5 +     module_dir_paths = sorted(set(
6 -         iree_artifacts.get_module_dir_path(config.module_generation_config)
6 +         str(register_access.get_module_dir_path(config.module_generation_config))
8 -     ])
8 +     ))
```

Figure 3: Examples of Variation in Difficulty.

Comment Length = 8
FYI, this will spam console when running ‘aaa‘.

Comment Length = 18
By the format string it looks like parameters shall be reversed. Type shall be 1st and exception 2nd

---

Comment Length = 23
I’m wondering if it’s useful to show the message from the exception in this debug message, at least in the case of IOException.

Figure 4: Examples of Different Comment Lengths.

Code Edit Distance = 9

```
1     OnConflictAction onconflict_action = ts_chunk_dispatch_get_on_conflict_action(dispatch);
2     ResultRelInfo *resrelinfo, *relinfo;
3 -   bool has_compressed_chunk = (chunk->fd.compressed_chunk_id != 0);
3 +   bool is_compressed = (chunk->fd.compressed_chunk_id != 0)
4     /* permissions NOT checked here; were checked at hypertable level */
5     if (check_enable_rls(chunk->table_id, InvalidOid, false) == RLS_ENABLED)
```

Code Review: Should ‘is_compressed‘ now be given by the chunk’s compression status flag? Otherwise it looks like we have different ways of determining compression status.

Code Edit Distance = 25

```
1     void Server_Card::resetState()
2     setPT(QString());
3     setAnnotation(QString());
4     setDoesntUntap(false);
5 -     setFaceDown(false);
6     }
7
8     QString Server_Card::setAttribute(CardAttribute attribute, const QString &avalue, bool allCards)
```

Code Review: this causes a major bug: cards have their state reset when moved between the battlefield and the deck, their facedown state is then checked afterwards to determine what event to show to other players. with this change moving a facedown card to your deck (unknown to unknown) will tell all your opponents (but not you) what card it was.

Code Edit Distance = 67

```
1     public unsafe LazyStringValue GetDocumentId(LazyStringValue key)
2                 if (index == -1)
3                     return null;
4
5 -               _tmpLazyStringInstance = _context.GetLazyString(key.Buffer, index);
5 +               return _context.GetLazyString(key.Buffer, index);
6 -               return _tmpLazyStringInstance;
7         }
8
9         // TODO unify if possible with AllowedPathsValidator
```

Code Review: Why do you store that in the temporary variable?

Figure 5: Examples of Different Code Edit Distances.

Code Element Ratio = 0
Should this really be a compile time error? The fact that is can be imported multiple times does not mean that it will be.

Code Element Ratio = 0.13
For all of the fuzz tests, does it make sense to have versions for ‘len_prefixed‘ both ‘true‘ and ‘false‘ ?

Code Element Ratio = 0.37
I think you’re missing a ‘flb_free(seq_index_str);‘ there.
Other than that, would you mind change that comparison to ‘if (tmp_key == NULL) {‘ instead? I’d really appreciate it.

Figure 6: Examples of Different Code Element Ratios.

Specification Ratio = 0.67

```
    void hpx_thread_buffer :: resize ( const std :: size_t num_threads,
    }

    void *hpx_thread_buffer :: get ( std :: size_t thread_num) const noexcept {
-       KOKKOS_ASSERT(thread_num < m_num_threads);
+       KOKKOS_EXPECTS(thread_num < m_num_threads);
        if (m_data == nullptr ) {
            return nullptr ;
        }
        return &m_data[thread_num * m_size_per_thread];
    }

    void *hpx_thread_buffer :: get_extra_space () const noexcept {
-       KOKKOS_ASSERT(m_extra_space > 0);
+       KOKKOS_EXPECTS(m_extra_space > 0);
        if (m_data == nullptr ) {
            return nullptr ;
        }
```

Code Review: This is fine but just pointing out there is also a ‘KOKKOS_EXPECTS‘ that was meant for checking preconditions

---

Specification Ratio = 37.60

```
    private String addNashornJavaScriptEngineIfNecessary( String cp) {
        }

        private boolean requiresNashornJavaScriptEngine () {
-           String version = System.getProperty("java . specification . version");
+           return getJavaVersion () >= 15; // Nashorn was removed in Java 15
-           if (version . startsWith ("1.")) {
-               version = version . substring (2);
-           }
-           return Integer . parseInt ( version ) >= 15; // Nashorn was removed in Java 15
        }

    }
```

Code Review: You can use ‘getJavaVersion()‘ here.

Figure 7: Examples of Different Specification Ratios.

Scale	Model	Parameters	Organisation	Release Date	Hugging Face
≤3B	Llama-3.2-3B-Instruct	3.2B	Meta	Sep 25, 2024	meta-llama/Llama-3.2-3B-Instruct
	Llama-3.2-1B-Instruct	1.2B	Meta	Sep 25, 2024	meta-llama/Llama-3.2-1B-Instruct
	Qwen2.5-Coder-3B-Instruct	3.1B	Alibaba	Nov 6, 2024	Qwen/Qwen2.5-Coder-3B-Instruct
	Qwen2.5-Coder-1.5B-Instruct	1.5B	Alibaba	Sep 18, 2024	Qwen/Qwen2.5-Coder-1.5B-Instruct
	Qwen2.5-3B-Instruct	3.1B	Alibaba	Sep 18, 2024	Qwen/Qwen2.5-3B-Instruct
	Qwen2.5-1.5B-Instruct	1.5B	Alibaba	Sep 18, 2024	Qwen/Qwen2.5-1.5B-Instruct
	deepseek-coder-1.3b-instruct	1.3B	DeepSeek	Oct 30, 2023	deepseek-ai/deepseek-coder-1.3b-instruct
	DeepSeek-R1-Distill-Qwen-1.5B	1.5B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
	Falcon3-3B-Instruct	3.2B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-3B-Instruct
	Falcon3-1B-Instruct	1.7B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-1B-Instruct
	Phi-3-mini-128k-instruct	3.8B	Microsoft	Apr 23, 2024	microsoft/Phi-3-mini-128k-instruct
	Yi-Coder-1.5B-Chat	1.5B	01.AI	Aug 27, 2024	01-ai/Yi-Coder-1.5B-Chat
	granite-3b-code-instruct-128k	3.5B	IBM	Jul 18, 2024	ibm-granite/granite-3b-code-instruct-128k
	granite-3.0-3b-a800m-instruct	3.4B	IBM	Oct 21, 2024	ibm-granite/granite-3.0-3b-a800m-instruct
	granite-3.0-2b-instruct	2.6B	IBM	Oct 21, 2024	ibm-granite/granite-3.0-2b-instruct
	EXAONE-3.5-2.4B-Instruct	2.4B	LG AI	Dec 9, 2024	LGAI-EXAONE/EXAONE-3.5-2.4B-Instruct
	internlm2_5-1_8b-chat	1.9B	InternLM	Jul 30, 2024	internlm/internlm2_5-1_8b-chat
	stable-code-instruct-3b	2.8B	Stability AI	Mar 19, 2024	stabilityai/stable-code-instruct-3b
≤9B	CodeLlama-7b-Instruct-hf	6.7B	Meta	Mar 13, 2024	meta-llama/CodeLlama-7b-Instruct-hf
	Llama-3.1-8B-Instruct	8.0B	Meta	Jul 18, 2024	meta-llama/Llama-3.1-8B-Instruct
	codegemma-1.1-7b-it	8.5B	Google	Apr 30, 2024	google/codegemma-1.1-7b-it
	gemma-2-9b-it	9.2B	Google	Jun 25, 2024	google/gemma-2-9b-it
	Qwen2.5-Coder-7B-Instruct	7.6B	Alibaba	Sep 17, 2024	Qwen/Qwen2.5-Coder-7B-Instruct
	Qwen2.5-7B-Instruct	7.6B	Alibaba	Sep 16, 2024	Qwen/Qwen2.5-7B-Instruct
	Marco-o1	7.6B	AIDC-AI	Nov 13, 2024	AIDC-AI/Marco-o1
	deepseek-coder-7b-instruct-v1.5	6.9B	DeepSeek	Jan 25, 2024	deepseek-ai/deepseek-coder-7b-instruct-v1.5
	deepseek-llm-7b-chat	6.9B	DeepSeek	Nov 29, 2023	deepseek-ai/deepseek-llm-7b-chat
	DeepSeek-R1-Distill-Qwen-7B	7.6B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-7B
	DeepSeek-R1-Distill-Llama-8B	8.0B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Llama-8B
	Falcon3-7B-Instruct	7.5B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-7B-Instruct
	Baichuan2-7B-Chat	7.1B	Baichuan AI	Sep 6, 2023	baichuan-inc/Baichuan2-7B-Chat
	Yi-Coder-9B-Chat	8.8B	01.AI	Aug 27, 2024	01-ai/Yi-Coder-9B-Chat
	Yi-1.5-9B-Chat	8.8B	01.AI	May 10, 2024	01-ai/Yi-1.5-9B-Chat
	granite-8b-code-instruct-128k	8.1B	IBM	Jul 12, 2024	ibm-granite/granite-8b-code-instruct-128k
	granite-3.0-8b-instruct	8.2B	IBM	Oct 15, 2024	ibm-granite/granite-3.0-8b-instruct
	EXAONE-3.5-7.8B-Instruct	7.8B	LG AI	Dec 9, 2024	LGAI-EXAONE/EXAONE-3.5-7.8B-Instruct
≤16B	CodeLlama-13b-Instruct-hf	13.0B	Meta	Mar 13, 2024	meta-llama/CodeLlama-13b-Instruct-hf
	Qwen2.5-Coder-14B-Instruct	14.8B	Alibaba	Nov 6, 2024	Qwen/Qwen2.5-Coder-14B-Instruct
	Qwen2.5-14B-Instruct	14.8B	Alibaba	Sep 16, 2024	Qwen/Qwen2.5-14B-Instruct
	DeepSeek-Coder-V2-Lite-Instruct	15.7B	DeepSeek	Jun 14, 2024	deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct
	DeepSeek-V2-Lite-Chat	15.7B	DeepSeek	May 15, 2024	deepseek-ai/DeepSeek-V2-Lite-Chat
	DeepSeek-R1-Distill-Qwen-14B	14.8B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
	falcon-11B	11.1B	TII UAE	May 9, 2024	tiiuae/falcon-11B
	Falcon3-10B-Instruct	10.3B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-10B-Instruct
	Baichuan2-13B-Chat	13.0B	Baichuan AI	Sep 6, 2023	baichuan-inc/Baichuan2-13B-Chat
	WizardLM-13B-V1.2	13.0B	WizardLM Team	Jul 25, 2023	WizardLMTeam/WizardLM-13B-V1.2
	starcoder2-15b-instruct-v0.1	16.0B	BigCode	Apr 23, 2024	bigcode/starcoder2-15b-instruct-v0.1
	Mistral-Nemo-Instruct-2407	12.2B	Mistral	Jul 17, 2024	mistralai/Mistral-Nemo-Instruct-2407
≤34B	CodeLlama-34b-Instruct-hf	33.7B	Meta	Mar 14, 2024	meta-llama/CodeLlama-34b-Instruct-hf
	gemma-2-27b-it	27.2B	Google	Jun 25, 2024	google/gemma-2-27b-it
	Qwen2.5-Coder-32B-Instruct	32.8B	Alibaba	Nov 6, 2024	Qwen/Qwen2.5-Coder-32B-Instruct
	Qwen2.5-32B-Instruct	32.8B	Alibaba	Sep 17, 2024	Qwen/Qwen2.5-32B-Instruct
	QwQ-32B-Preview	32.8B	Alibaba	Nov 27, 2024	Qwen/QwQ-32B-Preview
	Sky-T1-32B-Preview	32.8B	NovaSky	Jan 9, 2025	NovaSky-AI/Sky-T1-32B-Preview
	deepseek-coder-33b-instruct	33.3B	DeepSeek	Nov 1, 2023	deepseek-ai/deepseek-coder-33b-instruct
	DeepSeek-R1-Distill-Qwen-32B	32.8B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-32B
≤72B	CodeLlama-70b-Instruct-hf	69.0B	Meta	Mar 14, 2024	meta-llama/CodeLlama-70b-Instruct-hf
	Llama-3.1-70B-Instruct	70.6B	Meta	Jul 16, 2024	meta-llama/Llama-3.1-70B-Instruct
	Llama-3.3-70B-Instruct	70.6B	Meta	Nov 26, 2024	meta-llama/Llama-3.3-70B-Instruct
	Qwen2.5-72B-Instruct	72.7B	Alibaba	Sep 16, 2024	Qwen/Qwen2.5-72B-Instruct
	deepseek-llm-67b-chat	67.0B	DeepSeek	Nov 29, 2023	deepseek-ai/deepseek-llm-67b-chat
	DeepSeek-R1-Distill-Llama-70B	70.6B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Llama-70B
	WizardLM-70B-V1.0	70.0B	WizardLM Team	Aug 9, 2023	WizardLMTeam/WizardLM-70B-V1.0
	K2-Chat	65.3B	LLM360	May 28, 2024	LLM360/K2-Chat
	falcon-40b-instruct	40.0B	TII UAE	May 25, 2023	tiiuae/falcon-40b-instruct

Table 6: List of benchmarked models.

Scale	Model	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	23.0	18.0	22.0	21.0	29.0	24.0	36.0	33.0	27.0	25.9
	Llama-3.2-1B-Instruct	4.0	7.0	1.0	9.0	11.0	2.0	9.0	5.0	3.0	5.7
	Qwen2.5-Coder-3B-Instruct	24.0	28.0	27.0	30.0	30.0	25.0	42.0	30.0	37.0	30.3
	Qwen2.5-Coder-1.5B-Instruct	11.0	16.0	19.0	23.0	23.0	19.0	36.0	23.0	21.0	21.2
	Qwen2.5-3B-Instruct	15.0	16.0	12.0	24.0	18.0	26.0	31.0	24.0	26.0	21.3
	Qwen2.5-1.5B-Instruct	14.0	17.0	12.0	16.0	24.0	19.0	27.0	23.0	28.0	20.0
	deepseek-coder-1.3b-instruct	7.0	8.0	10.0	13.0	13.0	6.0	16.0	16.0	16.0	11.7
	DeepSeek-R1-Distill-Qwen-1.5B	0.0	5.0	3.0	4.0	1.0	2.0	2.0	2.0	1.0	2.2
	Falcon3-3B-Instruct	11.0	14.0	7.0	19.0	14.0	16.0	21.0	8.0	18.0	14.2
	Falcon3-1B-Instruct	2.0	3.0	2.0	7.0	5.0	4.0	4.0	2.0	2.0	3.4
	Phi-3-mini-128k-instruct	15.0	6.0	3.0	2.0	1.0	11.0	8.0	8.0	15.0	7.7
	Yi-Coder-1.5B-Chat	5.0	6.0	3.0	4.0	4.0	4.0	10.0	5.0	10.0	5.7
	granite-3b-code-instruct-128k	21.0	23.0	21.0	28.0	27.0	23.0	24.0	30.0	24.0	24.6
	granite-3.0-3b-a800m-instruct	10.0	14.0	6.0	13.0	21.0	11.0	18.0	23.0	16.0	14.7
	granite-3.0-2b-instruct	16.0	18.0	13.0	16.0	19.0	19.0	29.0	29.0	25.0	20.4
	EXAONE-3.5-2.4B-Instruct	5.0	6.0	1.0	0.0	2.0	4.0	4.0	3.0	9.0	3.8
	internlm2_5-1_8b-chat	5.0	5.0	2.0	3.0	6.0	4.0	7.0	7.0	4.0	4.8
	stable-code-instruct-3b	8.0	4.0	1.0	4.0	2.0	2.0	10.0	10.0	5.0	5.1
≤9B	CodeLlama-7b-Instruct-hf	30.0	27.0	27.0	36.0	41.0	30.0	43.0	40.0	41.0	35.0
	Llama-3.1-8B-Instruct	28.0	31.0	16.0	17.0	16.0	20.0	30.0	38.0	40.0	26.2
	codegemma-1.1-7b-it	22.0	10.0	25.0	32.0	21.0	32.0	26.0	27.0	41.0	26.2
	gemma-2-9b-it	29.0	31.0	44.0	39.0	34.0	39.0	46.0	47.0	42.0	39.0
	Qwen2.5-Coder-7B-Instruct	29.0	38.0	44.0	37.0	40.0	39.0	47.0	47.0	48.0	41.0
	Qwen2.5-7B-Instruct	22.0	21.0	33.0	34.0	41.0	36.0	43.0	43.0	48.0	35.7
	Marco-o1	27.0	28.0	33.0	37.0	34.0	32.0	40.0	41.0	43.0	35.0
	deepseek-coder-7b-instruct-v1.5	33.0	34.0	28.0	39.0	37.0	34.0	36.0	44.0	42.0	36.3
	deepseek-llm-7b-chat	15.0	15.0	14.0	17.0	21.0	14.0	20.0	26.0	18.0	17.8
	DeepSeek-R1-Distill-Qwen-7B	3.0	10.0	6.0	8.0	9.0	7.0	4.0	4.0	9.0	6.7
	DeepSeek-R1-Distill-Llama-8B	3.0	5.0	3.0	6.0	1.0	8.0	3.0	6.0	9.0	4.9
	Falcon3-7B-Instruct	16.0	15.0	7.0	14.0	12.0	14.0	19.0	11.0	21.0	14.3
	Baichuan2-7B-Chat	9.0	15.0	8.0	15.0	18.0	11.0	21.0	18.0	13.0	14.2
	Yi-Coder-9B-Chat	23.0	27.0	20.0	35.0	37.0	28.0	40.0	42.0	40.0	32.4
	Yi-1.5-9B-Chat	24.0	30.0	22.0	30.0	29.0	34.0	31.0	38.0	38.0	30.7
	granite-8b-code-instruct-128k	22.0	28.0	25.0	34.0	38.0	30.0	41.0	41.0	43.0	33.6
	granite-3.0-8b-instruct	23.0	24.0	30.0	29.0	31.0	27.0	43.0	37.0	42.0	31.8
	EXAONE-3.5-7.8B-Instruct	15.0	12.0	19.0	20.0	12.0	8.0	22.0	20.0	26.0	17.1
≤16B	CodeLlama-13b-Instruct-hf	29.0	32.0	30.0	37.0	43.0	36.0	43.0	42.0	38.0	36.7
	Qwen2.5-Coder-14B-Instruct	36.0	34.0	45.0	43.0	47.0	46.0	54.0	54.0	60.0	46.6
	Qwen2.5-14B-Instruct	26.0	31.0	29.0	34.0	30.0	38.0	49.0	42.0	52.0	36.8
	DeepSeek-Coder-V2-Lite-Instruct	20.0	29.0	10.0	23.0	33.0	25.0	38.0	30.0	35.0	27.0
	DeepSeek-V2-Lite-Chat	12.0	7.0	10.0	14.0	24.0	16.0	16.0	22.0	14.0	15.0
	DeepSeek-R1-Distill-Qwen-14B	25.0	27.0	20.0	32.0	28.0	29.0	46.0	41.0	34.0	31.3
	falcon-11B	16.0	14.0	22.0	23.0	19.0	17.0	23.0	23.0	20.0	19.7
	Falcon3-10B-Instruct	19.0	23.0	14.0	14.0	24.0	15.0	26.0	27.0	39.0	22.3
	Baichuan2-13B-Chat	11.0	14.0	12.0	17.0	10.0	13.0	24.0	25.0	19.0	16.1
	WizardLM-13B-V1.2	15.0	20.0	21.0	20.0	19.0	23.0	32.0	31.0	25.0	22.9
	Phi-3-medium-128k-instruct	21.0	22.0	24.0	32.0	35.0	29.0	35.0	38.0	36.0	30.2
	phi-4	41.0	23.0	38.0	36.0	38.0	37.0	37.0	48.0	36.0	37.1
	starcoder2-15b-instruct-v0.1	25.0	34.0	29.0	40.0	39.0	29.0	29.0	39.0	39.0	33.7
	Mistral-Nemo-Instruct-2407	29.0	22.0	27.0	33.0	29.0	38.0	38.0	39.0	43.0	33.1
≤34B	CodeLlama-34b-Instruct-hf	37.0	36.0	38.0	47.0	45.0	41.0	51.0	46.0	47.0	43.1
	gemma-2-27b-it	38.0	37.0	42.0	47.0	43.0	47.0	56.0	54.0	54.0	46.4
	Qwen2.5-Coder-32B-Instruct	35.0	35.0	40.0	45.0	40.0	39.0	51.0	51.0	61.0	44.1
	Qwen2.5-32B-Instruct	38.0	41.0	40.0	**52.0**	**49.0**	32.0	51.0	47.0	55.0	45.0
	QwQ-32B-Preview	39.0	42.0	**46.0**	51.0	45.0	43.0	**58.0**	42.0	44.0	45.6
	Sky-T1-32B-Preview	36.0	38.0	35.0	49.0	43.0	32.0	52.0	46.0	51.0	42.4
	deepseek-coder-33b-instruct	28.0	30.0	22.0	40.0	32.0	28.0	38.0	40.0	37.0	32.8
	DeepSeek-R1-Distill-Qwen-32B	25.0	23.0	36.0	22.0	25.0	31.0	36.0	35.0	42.0	30.6
	Yi-1.5-34B-Chat	22.0	30.0	28.0	28.0	27.0	27.0	39.0	39.0	40.0	31.1
	Mistral-Small-Instruct-2409	34.0	39.0	44.0	42.0	42.0	51.0	53.0	50.0	55.0	45.6
	granite-34b-code-instruct-8k	23.0	27.0	31.0	36.0	35.0	32.0	48.0	45.0	42.0	35.4
	internlm2_5-20b-chat	29.0	24.0	31.0	37.0	33.0	28.0	40.0	40.0	39.0	33.4
	EXAONE-3.5-32B-Instruct	27.0	28.0	28.0	24.0	28.0	29.0	39.0	32.0	39.0	30.4
≤72B	CodeLlama-70b-Instruct-hf	42.0	38.0	38.0	48.0	48.0	41.0	54.0	53.0	45.0	45.2
	Llama-3.1-70B-Instruct	**46.0**	42.0	45.0	49.0	47.0	**57.0**	55.0	50.0	**62.0**	**50.3**
	Llama-3.3-70B-Instruct	40.0	33.0	36.0	19.0	31.0	34.0	40.0	17.0	46.0	32.9
	Qwen2.5-72B-Instruct	**46.0**	**45.0**	44.0	44.0	48.0	44.0	53.0	**56.0**	58.0	48.7
	deepseek-llm-67b-chat	29.0	33.0	31.0	41.0	35.0	43.0	50.0	44.0	51.0	39.7
	DeepSeek-R1-Distill-Llama-70B	20.0	23.0	23.0	23.0	26.0	43.0	35.0	24.0	41.0	28.7
	WizardLM-70B-V1.0	16.0	31.0	29.0	28.0	32.0	33.0	36.0	32.0	41.0	30.9
	K2-Chat	34.0	35.0	36.0	41.0	35.0	33.0	50.0	48.0	47.0	39.9
	falcon-40b-instruct	17.0	13.0	12.0	20.0	21.0	21.0	27.0	24.0	16.0	19.0

Best Score Overall, Best Score within Scale

Table 7: Automated Code Refinement Results (Exact Match Rate).

Scale	Model	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	73.0	80.0	73.0	80.0	82.0	79.0	85.0	72.0	85.0	78.8
	Llama-3.2-1B-Instruct	72.0	76.0	72.0	77.0	79.0	77.0	80.0	73.0	79.0	76.1
	Qwen2.5-Coder-3B-Instruct	72.0	76.0	72.0	80.0	81.0	80.0	81.0	71.0	86.0	77.7
	Qwen2.5-Coder-1.5B-Instruct	67.0	74.0	67.0	75.0	76.0	73.0	76.0	65.0	81.0	72.7
	Qwen2.5-3B-Instruct	70.0	76.0	64.0	72.0	72.0	64.0	68.0	59.0	74.0	68.8
	Qwen2.5-1.5B-Instruct	71.0	79.0	70.0	80.0	83.0	78.0	82.0	74.0	87.0	78.2
	deepseek-coder-1.3b-instruct	11.0	0.0	1.0	1.0	3.0	0.0	0.0	0.0	1.0	1.9
	DeepSeek-R1-Distill-Qwen-1.5B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	42.0	58.0	50.0	53.0	53.0	56.0	59.0	43.0	54.0	52.0
	Falcon3-1B-Instruct	36.0	64.0	33.0	66.0	62.0	52.0	66.0	53.0	70.0	55.8
	Phi-3-mini-128k-instruct	73.0	77.0	74.0	74.0	81.0	77.0	81.0	72.0	83.0	76.9
	Yi-Coder-1.5B-Chat	0.0	5.0	5.0	25.0	3.0	2.0	5.0	7.0	9.0	6.8
	granite-3b-code-instruct-128k	28.0	23.0	26.0	40.0	36.0	30.0	21.0	22.0	26.0	28.0
	granite-3.0-3b-a800m-instruct	50.0	37.0	28.0	37.0	17.0	19.0	40.0	23.0	17.0	29.8
	granite-3.0-2b-instruct	66.0	69.0	69.0	70.0	76.0	69.0	72.0	54.0	76.0	69.0
	EXAONE-3.5-2.4B-Instruct	55.0	71.0	65.0	67.0	69.0	58.0	73.0	52.0	78.0	65.3
	internlm2_5-1_8b-chat	70.0	69.0	58.0	68.0	67.0	52.0	79.0	60.0	57.0	64.4
	stable-code-instruct-3b	0.0	0.0	0.0	0.0	3.0	2.0	1.0	0.0	2.0	0.9
≤9B	CodeLlama-7b-Instruct-hf	68.0	78.0	73.0	78.0	78.0	75.0	81.0	66.0	82.0	75.4
	Llama-3.1-8B-Instruct	68.0	74.0	65.0	77.0	72.0	74.0	77.0	62.0	77.0	71.8
	codegemma-1.1-7b-it	73.0	80.0	75.0	81.0	83.0	80.0	86.0	74.0	87.0	79.9
	gemma-2-9b-it	72.0	80.0	69.0	74.0	78.0	75.0	76.0	59.0	84.0	74.1
	Qwen2.5-Coder-7B-Instruct	71.0	80.0	73.0	80.0	83.0	80.0	85.0	70.0	85.0	78.6
	Qwen2.5-7B-Instruct	72.0	79.0	74.0	79.0	81.0	74.0	70.0	71.0	83.0	75.9
	Marco-o1	73.0	81.0	76.0	81.0	82.0	74.0	76.0	75.0	85.0	78.1
	deepseek-coder-7b-instruct-v1.5	58.0	68.0	58.0	65.0	54.0	51.0	61.0	44.0	64.0	58.1
	deepseek-llm-7b-chat	66.0	76.0	66.0	75.0	78.0	74.0	80.0	66.0	79.0	73.3
	DeepSeek-R1-Distill-Qwen-7B	1.0	1.0	0.0	0.0	1.0	2.0	0.0	1.0	3.0	1.0
	DeepSeek-R1-Distill-Llama-8B	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.4
	Falcon3-7B-Instruct	66.0	76.0	73.0	75.0	79.0	73.0	74.0	67.0	76.0	73.2
	Baichuan2-7B-Chat	25.0	77.0	74.0	80.0	77.0	78.0	84.0	68.0	82.0	71.7
	Yi-Coder-9B-Chat	57.0	66.0	44.0	54.0	50.0	50.0	63.0	39.0	46.0	52.1
	Yi-1.5-9B-Chat	73.0	80.0	75.0	83.0	83.0	77.0	79.0	72.0	85.0	78.6
	granite-8b-code-instruct-128k	71.0	78.0	71.0	77.0	77.0	76.0	81.0	69.0	82.0	75.8
	granite-3.0-8b-instruct	56.0	62.0	47.0	57.0	51.0	53.0	68.0	45.0	64.0	55.9
	EXAONE-3.5-7.8B-Instruct	67.0	66.0	63.0	70.0	65.0	66.0	74.0	60.0	77.0	67.6
≤16B	CodeLlama-13b-Instruct-hf	42.0	78.0	68.0	68.0	66.0	74.0	77.0	58.0	79.0	67.8
	Qwen2.5-Coder-14B-Instruct	69.0	79.0	66.0	74.0	77.0	74.0	82.0	64.0	80.0	73.9
	Qwen2.5-14B-Instruct	68.0	73.0	76.0	74.0	80.0	73.0	77.0	62.0	77.0	73.3
	DeepSeek-Coder-V2-Lite-Instruct	70.0	81.0	77.0	80.0	79.0	69.0	75.0	72.0	78.0	75.7
	DeepSeek-V2-Lite-Chat	72.0	80.0	66.0	79.0	83.0	77.0	84.0	69.0	81.0	76.8
	DeepSeek-R1-Distill-Qwen-14B	67.0	68.0	72.0	72.0	77.0	76.0	78.0	64.0	80.0	72.7
	falcon-11B	72.0	80.0	74.0	81.0	83.0	80.0	85.0	74.0	87.0	79.6
	Falcon3-10B-Instruct	71.0	54.0	53.0	44.0	50.0	26.0	51.0	31.0	22.0	44.7
	Baichuan2-13B-Chat	14.0	80.0	67.0	76.0	78.0	75.0	83.0	69.0	76.0	68.7
	WizardLM-13B-V1.2	42.0	69.0	66.0	68.0	70.0	51.0	74.0	67.0	79.0	65.1
	starcoder2-15b-instruct-v0.1	48.0	43.0	31.0	42.0	38.0	43.0	50.0	37.0	29.0	40.1
	Mistral-Nemo-Instruct-2407	69.0	78.0	73.0	80.0	76.0	76.0	78.0	69.0	83.0	75.8
≤34B	CodeLlama-34b-Instruct-hf	1.0	4.0	1.0	0.0	1.0	0.0	2.0	2.0	3.0	1.6
	gemma-2-27b-it	69.0	77.0	70.0	76.0	73.0	73.0	77.0	68.0	83.0	74.0
	Qwen2.5-Coder-32B-Instruct	69.0	82.0	81.0	82.0	85.0	77.0	80.0	72.0	84.0	79.1
	Qwen2.5-32B-Instruct	73.0	76.0	71.0	68.0	78.0	72.0	64.0	67.0	70.0	71.0
	QwQ-32B-Preview	47.0	64.0	59.0	60.0	66.0	62.0	63.0	55.0	67.0	60.3
	Sky-T1-32B-Preview	56.0	68.0	61.0	58.0	71.0	64.0	60.0	61.0	64.0	62.6
	deepseek-coder-33b-instruct	42.0	70.0	60.0	76.0	74.0	72.0	78.0	71.0	85.0	69.8
	DeepSeek-R1-Distill-Qwen-32B	69.0	77.0	76.0	74.0	80.0	72.0	72.0	63.0	76.0	73.2
≤72B	CodeLlama-70b-Instruct-hf	72.0	79.0	71.0	81.0	83.0	75.0	83.0	73.0	83.0	77.8
	Llama-3.1-70B-Instruct	59.0	71.0	75.0	68.0	71.0	74.0	67.0	63.0	68.0	68.4
	Llama-3.3-70B-Instruct	59.0	68.0	69.0	67.0	70.0	73.0	65.0	59.0	64.0	66.0
	Qwen2.5-72B-Instruct	75.0	81.0	82.0	80.0	83.0	79.0	77.0	75.0	86.0	79.8
	deepseek-llm-67b-chat	74.0	81.0	73.0	81.0	82.0	79.0	84.0	72.0	86.0	79.1
	DeepSeek-R1-Distill-Llama-70B	50.0	68.0	63.0	55.0	67.0	65.0	60.0	60.0	63.0	61.2
	WizardLM-70B-V1.0	72.0	82.0	73.0	80.0	80.0	81.0	82.0	73.0	84.0	78.6
	K2-Chat	72.0	79.0	71.0	82.0	83.0	77.0	81.0	72.0	86.0	78.1
	falcon-40b-instruct	12.0	57.0	58.0	72.0	69.0	65.0	74.0	57.0	77.0	60.1

Best Score Overall, Best Score within Scale

Table 8: Change Type Recognition (Invariant Accuracy).

Scale	Model	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	2.0	0.0	0.0	0.0	1.0	0.0	2.0	1.0	1.0	0.8
	Llama-3.2-1B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-Coder-3B-Instruct	0.0	4.0	1.0	2.0	2.0	0.0	2.0	2.0	3.0	1.8
	Qwen2.5-Coder-1.5B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-3B-Instruct	41.0	34.0	45.0	39.0	45.0	31.0	46.0	37.0	36.0	39.3
	Qwen2.5-1.5B-Instruct	2.0	1.0	2.0	1.0	3.0	1.0	4.0	1.0	1.0	1.8
	deepseek-coder-1.3b-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DeepSeek-R1-Distill-Qwen-1.5B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	0.0	3.0	1.0	5.0	2.0	1.0	2.0	0.0	1.0	1.7
	Falcon3-1B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	27.0	24.0	34.0	38.0	32.0	28.0	48.0	43.0	33.0	34.1
	Yi-Coder-1.5B-Chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3b-code-instruct-128k	0.0	1.0	0.0	1.0	1.0	1.0	2.0	1.0	0.0	0.8
	granite-3.0-3b-a800m-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3.0-2b-instruct	4.0	2.0	3.0	1.0	1.0	1.0	4.0	0.0	0.0	1.8
	EXAONE-3.5-2.4B-Instruct	3.0	1.0	1.0	0.0	0.0	5.0	1.0	2.0	0.0	1.4
	internlm2_5-1_8b-chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	stable-code-instruct-3b	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
≤9B	CodeLlama-7b-Instruct-hf	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.3
	Llama-3.1-8B-Instruct	37.0	39.0	39.0	37.0	38.0	28.0	48.0	41.0	28.0	37.2
	codegemma-1.1-7b-it	5.0	1.0	4.0	6.0	6.0	1.0	3.0	3.0	3.0	3.6
	gemma-2-9b-it	45.0	55.0	55.0	68.0	58.0	61.0	72.0	60.0	59.0	59.2
	Qwen2.5-Coder-7B-Instruct	6.0	7.0	16.0	14.0	13.0	9.0	38.0	12.0	9.0	13.8
	Qwen2.5-7B-Instruct	31.0	27.0	39.0	37.0	37.0	36.0	17.0	28.0	34.0	31.8
	Marco-o1	31.0	26.0	36.0	36.0	27.0	38.0	40.0	28.0	28.0	32.2
	deepseek-coder-7b-instruct-v1.5	7.0	4.0	9.0	4.0	8.0	5.0	9.0	2.0	3.0	5.7
	deepseek-llm-7b-chat	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	4.0	0.6
	DeepSeek-R1-Distill-Qwen-7B	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.2
	DeepSeek-R1-Distill-Llama-8B	1.0	0.0	0.0	0.0	2.0	0.0	1.0	0.0	2.0	0.7
	Falcon3-7B-Instruct	18.0	25.0	28.0	30.0	37.0	26.0	43.0	30.0	25.0	29.1
	Baichuan2-7B-Chat	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.1
	Yi-Coder-9B-Chat	2.0	3.0	9.0	10.0	8.0	4.0	8.0	8.0	1.0	5.9
	Yi-1.5-9B-Chat	33.0	42.0	37.0	44.0	43.0	40.0	51.0	42.0	40.0	41.3
	granite-8b-code-instruct-128k	12.0	13.0	19.0	19.0	24.0	14.0	18.0	16.0	9.0	16.0
	granite-3.0-8b-instruct	20.0	15.0	28.0	35.0	27.0	28.0	41.0	29.0	17.0	26.7
	EXAONE-3.5-7.8B-Instruct	28.0	21.0	32.0	48.0	38.0	33.0	40.0	32.0	29.0	33.4
≤16B	CodeLlama-13b-Instruct-hf	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.11
	Qwen2.5-Coder-14B-Instruct	42.0	31.0	49.0	49.0	49.0	50.0	61.0	47.0	42.0	46.7
	Qwen2.5-14B-Instruct	60.0	59.0	64.0	72.0	60.0	74.0	74.0	60.0	66.0	65.4
	DeepSeek-Coder-V2-Lite-Instruct	11.0	6.0	6.0	7.0	9.0	7.0	8.0	4.0	4.0	6.9
	DeepSeek-V2-Lite-Chat	2.0	3.0	10.0	9.0	11.0	7.0	13.0	5.0	7.0	7.4
	DeepSeek-R1-Distill-Qwen-14B	46.0	47.0	62.0	59.0	58.0	59.0	68.0	61.0	62.0	58.0
	falcon-11B	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	Falcon3-10B-Instruct	9.0	16.0	18.0	17.0	21.0	10.0	27.0	14.0	8.0	15.6
	Baichuan2-13B-Chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	WizardLM-13B-V1.2	1.0	0.0	0.0	0.0	1.0	0.0	2.0	0.0	0.0	0.4
	starcoder2-15b-instruct-v0.1	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.3
	Mistral-Nemo-Instruct-2407	25.0	24.0	35.0	36.0	32.0	33.0	43.0	35.0	31.0	32.7
≤34B	CodeLlama-34b-Instruct-hf	1.0	0.0	3.0	3.0	3.0	4.0	0.0	0.0	2.0	1.8
	gemma-2-27b-it	64.0	62.0	79.0	71.0	68.0	72.0	76.0	72.0	67.0	70.1
	Qwen2.5-Coder-32B-Instruct	64.0	63.0	74.0	74.0	74.0	76.0	76.0	71.0	72.0	71.6
	Qwen2.5-32B-Instruct	58.0	58.0	76.0	81.0	72.0	77.0	77.0	70.0	69.0	70.9
	QwQ-32B-Preview	39.0	41.0	61.0	54.0	49.0	60.0	65.0	53.0	47.0	52.1
	Sky-T1-32B-Preview	47.0	49.0	66.0	64.0	56.0	66.0	70.0	58.0	58.0	59.3
	deepseek-coder-33b-instruct	0.0	3.0	3.0	6.0	2.0	3.0	8.0	10.0	3.0	4.2
	DeepSeek-R1-Distill-Qwen-32B	56.0	60.0	77.0	68.0	66.0	74.0	74.0	70.0	69.0	68.2
≤72B	CodeLlama-70b-Instruct-hf	8.0	9.0	14.0	10.0	13.0	17.0	31.0	16.0	11.0	14.3
	Llama-3.1-70B-Instruct	72.0	68.0	74.0	81.0	73.0	78.0	82.0	74.0	70.0	74.7
	Llama-3.3-70B-Instruct	65.0	68.0	76.0	80.0	76.0	78.0	82.0	73.0	72.0	74.4
	Qwen2.5-72B-Instruct	46.0	53.0	71.0	74.0	64.0	69.0	73.0	63.0	65.0	64.2
	deepseek-llm-67b-chat	36.0	28.0	47.0	44.0	49.0	38.0	60.0	45.0	35.0	42.4
	DeepSeek-R1-Distill-Llama-70B	45.0	44.0	62.0	51.0	53.0	57.0	71.0	55.0	57.0	55.0
	WizardLM-70B-V1.0	25.0	16.0	25.0	24.0	30.0	19.0	30.0	29.0	22.0	24.4
	K2-Chat	39.0	31.0	45.0	45.0	43.0	37.0	59.0	50.0	31.0	42.2
	falcon-40b-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Best Score Overall, Best Score within Scale

Table 9: Change Localisation - Easy (Invariant Accuracy).

Scale	Model	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.3
	Llama-3.2-1B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-Coder-3B-Instruct	0.0	3.0	0.0	2.0	2.0	1.0	3.0	2.0	3.0	1.8
	Qwen2.5-Coder-1.5B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-3B-Instruct	34.0	27.0	39.0	31.0	39.0	28.0	45.0	34.0	38.0	35.0
	Qwen2.5-1.5B-Instruct	0.0	1.0	1.0	1.0	2.0	1.0	4.0	0.0	1.0	1.2
	deepseek-coder-1.3b-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DeepSeek-R1-Distill-Qwen-1.5B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	0.0	3.0	1.0	5.0	2.0	1.0	2.0	0.0	1.0	1.7
	Falcon3-1B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	21.0	21.0	29.0	36.0	26.0	27.0	48.0	36.0	32.0	30.7
	Yi-Coder-1.5B-Chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3b-code-instruct-128k	0.0	1.0	0.0	1.0	0.0	1.0	2.0	1.0	0.0	0.7
	granite-3.0-3b-a800m-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3.0-2b-instruct	2.0	2.0	1.0	0.0	0.0	1.0	3.0	0.0	0.0	1.0
	EXAONE-3.5-2.4B-Instruct	1.0	0.0	1.0	0.0	0.0	3.0	1.0	1.0	0.0	0.8
	internlm2_5-1_8b-chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	stable-code-instruct-3b	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
≤9B	CodeLlama-7b-Instruct-hf	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.3
	Llama-3.1-8B-Instruct	27.0	30.0	29.0	27.0	29.0	28.0	41.0	32.0	25.0	29.8
	codegemma-1.1-7b-it	3.0	1.0	4.0	5.0	5.0	1.0	3.0	3.0	3.0	3.1
	gemma-2-9b-it	41.0	45.0	49.0	58.0	49.0	55.0	68.0	50.0	53.0	52.0
	Qwen2.5-Coder-7B-Instruct	6.0	5.0	15.0	13.0	12.0	8.0	18.0	10.0	9.0	10.7
	Qwen2.5-7B-Instruct	26.0	28.0	41.0	32.0	36.0	40.0	43.0	28.0	34.0	34.2
	Marco-o1	25.0	26.0	30.0	27.0	31.0	34.0	40.0	24.0	30.0	29.7
	deepseek-coder-7b-instruct-v1.5	5.0	4.0	6.0	2.0	5.0	5.0	9.0	2.0	3.0	4.6
	deepseek-llm-7b-chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.4
	DeepSeek-R1-Distill-Qwen-7B	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.2
	DeepSeek-R1-Distill-Llama-8B	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	2.0	0.6
	Falcon3-7B-Instruct	10.0	20.0	21.0	21.0	26.0	20.0	38.0	25.0	22.0	22.6
	Baichuan2-7B-Chat	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.1
	Yi-Coder-9B-Chat	1.0	1.0	4.0	7.0	4.0	5.0	8.0	4.0	1.0	3.9
	Yi-1.5-9B-Chat	36.0	46.0	40.0	44.0	45.0	43.0	54.0	45.0	36.0	43.2
	granite-8b-code-instruct-128k	10.0	9.0	16.0	15.0	17.0	11.0	15.0	10.0	11.0	12.7
	granite-3.0-8b-instruct	15.0	11.0	23.0	27.0	22.0	27.0	40.0	23.0	16.0	22.7
	EXAONE-3.5-7.8B-Instruct	18.0	18.0	25.0	35.0	29.0	29.0	37.0	28.0	27.0	27.3
≤16B	CodeLlama-13b-Instruct-hf	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	Qwen2.5-Coder-14B-Instruct	29.0	22.0	37.0	38.0	37.0	45.0	56.0	36.0	36.0	37.3
	Qwen2.5-14B-Instruct	51.0	48.0	58.0	59.0	55.0	67.0	69.0	46.0	61.0	57.1
	DeepSeek-Coder-V2-Lite-Instruct	9.0	4.0	3.0	7.0	8.0	7.0	8.0	4.0	6.0	6.2
	DeepSeek-V2-Lite-Chat	2.0	3.0	6.0	10.0	9.0	7.0	14.0	4.0	7.0	6.9
	DeepSeek-R1-Distill-Qwen-14B	43.0	39.0	57.0	51.0	52.0	57.0	67.0	50.0	63.0	53.2
	falcon-11B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-10B-Instruct	6.0	14.0	15.0	17.0	11.0	8.0	20.0	9.0	8.0	12.0
	Baichuan2-13B-Chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	WizardLM-13B-V1.2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.1
	starcoder2-15b-instruct-v0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Mistral-Nemo-Instruct-2407	19.0	23.0	32.0	32.0	29.0	31.0	40.0	25.0	30.0	29.0
≤34B	CodeLlama-34b-Instruct-hf	1.0	0.0	3.0	3.0	3.0	4.0	0.0	0.0	2.0	1.8
	gemma-2-27b-it	52.0	48.0	67.0	52.0	57.0	63.0	67.0	61.0	61.0	58.7
	Qwen2.5-Coder-32B-Instruct	53.0	55.0	66.0	66.0	64.0	74.0	76.0	56.0	69.0	64.3
	Qwen2.5-32B-Instruct	58.0	58.0	70.0	75.0	66.0	74.0	77.0	61.0	69.0	67.6
	QwQ-32B-Preview	37.0	43.0	52.0	54.0	44.0	62.0	63.0	47.0	49.0	50.1
	Sky-T1-32B-Preview	41.0	48.0	60.0	63.0	53.0	68.0	72.0	49.0	57.0	56.8
	deepseek-coder-33b-instruct	0.0	3.0	3.0	4.0	1.0	1.0	8.0	6.0	3.0	3.2
	DeepSeek-R1-Distill-Qwen-32B	54.0	58.0	65.0	67.0	61.0	74.0	73.0	61.0	68.0	64.6
≤72B	CodeLlama-70b-Instruct-hf	5.0	6.0	11.0	9.0	8.0	16.0	27.0	9.0	11.0	11.3
	Llama-3.1-70B-Instruct	65.0	61.0	67.0	73.0	68.0	76.0	80.0	66.0	65.0	69.0
	Llama-3.3-70B-Instruct	57.0	58.0	65.0	71.0	65.0	77.0	77.0	61.0	68.0	66.6
	Qwen2.5-72B-Instruct	42.0	47.0	60.0	62.0	61.0	65.0	71.0	50.0	67.0	58.3
	deepseek-llm-67b-chat	28.0	26.0	40.0	40.0	41.0	36.0	58.0	40.0	31.0	37.8
	DeepSeek-R1-Distill-Llama-70B	37.0	38.0	45.0	42.0	46.0	52.0	67.0	46.0	53.0	47.3
	WizardLM-70B-V1.0	22.0	9.0	18.0	20.0	21.0	18.0	25.0	20.0	20.0	19.2
	K2-Chat	29.0	29.0	40.0	47.0	38.0	37.0	56.0	47.0	32.0	39.4
	falcon-40b-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Best Score Overall, Best Score within Scale

Table 10: Change Localisation - Hard (Invariant Accuracy).

Scale	Model	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	3.75	5.88	11.69	16.28	3.49	6.98	7.78	8.97	23.86	9.9
	Llama-3.2-1B-Instruct	2.5	1.2	1.3	3.5	1.2	2.3	1.1	5.1	1.1	2.2
	Qwen2.5-Coder-3B-Instruct	11.25	11.8	15.6	11.6	17.4	6.9	16.7	15.4	3.4	12.2
	Qwen2.5-Coder-1.5B-Instruct	5.0	5.9	5.2	2.3	2.3	4.7	10.0	10.3	5.7	5.7
	Qwen2.5-3B-Instruct	50.0	61.2	52.0	53.5	59.3	56.9	62.2	51.3	51.1	55.3
	Qwen2.5-1.5B-Instruct	32.5	52.9	46.8	40.7	44.2	44.2	57.8	39.7	52.3	45.7
	deepseek-coder-1.3b-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.3	0.0	0.1
	DeepSeek-R1-Distill-Qwen-1.5B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	15.0	30.6	27.3	29.0	29.1	29.1	37.8	32.1	21.6	27.9
	Falcon3-1B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	53.75	55.3	59.7	54.7	62.8	69.8	56.7	55.1	55.7	58.2
	Yi-Coder-1.5B-Chat	0.0	1.2	0.0	1.2	0.0	0.0	0.0	0.0	0.0	0.3
	granite-3b-code-instruct-128k	3.75	2.4	0.0	4.7	2.3	0.0	3.3	6.4	3.4	2.9
	granite-3.0-3b-a800m-instruct	7.5	8.2	6.5	5.8	9.3	8.1	11.1	5.1	3.4	7.2
	granite-3.0-2b-instruct	20.0	21.1	14.3	22.1	18.6	22.1	21.1	23.1	9.1	19.1
	EXAONE-3.5-2.4B-Instruct	20.0	40.0	40.3	31.4	36.1	30.2	44.4	41.0	25.0	34.3
	internlm2_5-1_8b-chat	1.3	0.0	0.0	8.1	5.8	5.8	3.3	5.1	1.1	3.4
	stable-code-instruct-3b	1.3	0.0	2.6	1.2	0.0	3.5	2.2	2.6	0.0	1.5
≤9B	CodeLlama-7b-Instruct-hf	3.8	2.4	3.9	3.5	4.7	7.0	11.1	12.8	3.4	5.8
	Llama-3.1-8B-Instruct	57.5	64.7	70.1	67.4	67.4	68.6	68.9	67.9	58.0	65.6
	codegemma-1.1-7b-it	20.0	23.5	20.8	17.4	19.8	20.9	16.7	24.4	14.8	19.8
	gemma-2-9b-it	57.5	57.6	68.8	54.7	48.8	58.1	58.9	66.7	58.0	58.8
	Qwen2.5-Coder-7B-Instruct	58.8	62.4	75.3	66.3	73.3	72.1	68.9	65.4	65.9	67.6
	Qwen2.5-7B-Instruct	66.3	57.6	66.2	60.5	65.1	62.8	62.2	64.1	61.4	62.9
	Marco-o1	62.5	64.7	70.1	66.3	75.6	70.9	73.3	67.9	72.7	69.3
	deepseek-coder-7b-instruct-v1.5	15.0	10.6	11.7	20.9	15.1	14.0	18.9	20.5	12.5	15.5
	deepseek-llm-7b-chat	7.5	8.2	9.1	12.8	12.8	12.8	7.8	15.4	9.1	10.6
	DeepSeek-R1-Distill-Qwen-7B	0.0	0.0	0.0	1.2	0.0	0.0	0.0	0.0	0.0	0.1
	DeepSeek-R1-Distill-Llama-8B	13.8	7.1	13.0	15.1	9.3	9.3	13.3	14.1	13.6	12.1
	Falcon3-7B-Instruct	50.0	54.1	42.9	47.7	50.0	36.0	40.0	39.7	38.6	44.3
	Baichuan2-7B-Chat	0.0	2.4	1.3	5.8	2.3	2.3	3.3	6.4	5.7	3.3
	Yi-Coder-9B-Chat	40.0	43.5	50.6	39.5	33.7	39.5	43.3	44.9	33.0	40.9
	Yi-1.5-9B-Chat	60.0	68.2	64.9	59.3	70.9	72.1	61.1	71.8	65.9	66.0
	granite-8b-code-instruct-128k	15.0	25.9	15.6	24.4	16.3	19.8	23.3	30.8	29.5	22.3
	granite-3.0-8b-instruct	40.0	47.1	44.2	43.0	40.7	41.9	50.0	29.5	29.5	40.6
	EXAONE-3.5-7.8B-Instruct	60.0	69.4	71.4	65.1	68.6	72.1	76.7	66.7	65.9	68.4
≤16B	CodeLlama-13b-Instruct-hf	2.5	8.2	13.0	11.6	11.6	15.1	14.4	23.1	25.0	13.8
	Qwen2.5-Coder-14B-Instruct	62.5	55.3	72.7	66.3	75.6	61.6	67.8	61.5	65.9	65.5
	Qwen2.5-14B-Instruct	90.0	83.5	90.9	89.5	88.4	88.4	87.8	83.3	88.6	87.8
	DeepSeek-Coder-V2-Lite-Instruct	22.5	17.6	13.0	12.8	17.4	15.1	14.4	24.4	13.6	16.8
	DeepSeek-V2-Lite-Chat	27.5	34.1	28.6	26.0	23.3	20.9	24.4	28.2	18.2	25.7
	DeepSeek-R1-Distill-Qwen-14B	72.5	72.9	88.3	77.9	77.9	83.7	80.0	66.7	75.0	77.2
	falcon-11B	8.8	10.6	9.1	10.5	2.3	10.5	10.0	16.7	5.7	9.3
	Falcon3-10B-Instruct	42.5	32.9	23.4	18.6	23.3	17.4	30.0	11.5	14.8	23.8
	Baichuan2-13B-Chat	0.0	0.0	1.3	0.0	0.0	2.3	1.1	5.1	0.0	1.1
	WizardLM-13B-V1.2	6.3	5.9	2.6	11.6	1.2	9.3	5.6	9.0	2.3	6.0
	starcoder2-15b-instruct-v0.1	11.3	14.1	20.8	15.1	11.6	14.0	23.3	25.6	18.2	17.1
	Mistral-Nemo-Instruct-2407	57.5	58.8	67.5	62.8	65.1	55.8	57.8	61.5	65.9	61.4
≤34B	CodeLlama-34b-Instruct-hf	7.5	11.8	18.2	12.8	15.1	17.4	10.0	11.5	18.2	13.6
	gemma-2-27b-it	80.0	78.8	79.2	72.1	83.7	72.1	72.2	70.5	77.3	76.2
	Qwen2.5-Coder-32B-Instruct	91.3	84.7	96.1	94.2	91.9	96.5	94.4	85.9	89.8	91.6
	Qwen2.5-32B-Instruct	90.0	91.8	96.1	89.5	89.5	91.9	96.7	89.7	95.5	92.3
	QwQ-32B-Preview	68.8	72.9	89.6	74.4	80.2	84.9	82.2	74.4	84.1	79.1
	Sky-T1-32B-Preview	80.0	84.7	93.5	82.6	86.0	89.5	94.4	80.8	92.0	87.1
	deepseek-coder-33b-instruct	6.3	3.5	6.5	12.8	4.7	11.6	15.6	16.7	13.6	10.1
≤72B	CodeLlama-70b-Instruct-hf	32.5	38.8	42.9	37.2	34.9	46.5	42.2	43.6	38.6	39.7
	Llama-3.1-70B-Instruct	85.0	77.6	88.3	83.7	87.2	90.7	86.7	74.4	84.1	84.2
	Llama-3.3-70B-Instruct	86.3	76.5	89.6	83.7	83.7	90.7	87.8	75.6	87.5	84.6
	Qwen2.5-72B-Instruct	97.5	98.8	97.4	96.5	98.8	97.7	97.8	93.6	95.5	97.1
	deepseek-llm-67b-chat	55.0	49.4	50.6	58.1	50.0	57.0	55.6	55.1	59.1	54.4
	DeepSeek-R1-Distill-Llama-70B	71.3	77.6	74.0	81.4	79.1	84.9	80.0	76.9	76.1	77.9
	WizardLM-70B-V1.0	52.5	52.9	62.3	55.8	60.5	61.6	62.2	56.4	55.7	57.8
	K2-Chat	73.8	64.7	90.9	72.1	79.1	72.1	77.8	75.6	64.8	74.5
	falcon-40b-instruct	2.5	5.9	10.4	9.3	2.3	8.1	7.8	12.8	1.1	6.7

Best Score Overall, Best Score within Scale

Table 11: Solution Identification - Easy (Invariant Accuracy).

Scale	Model	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	5.0	4.7	7.8	16.3	1.2	7.0	6.7	7.7	12.5	7.6
	Llama-3.2-1B-Instruct	0.0	1.2	0.0	1.2	0.0	0.0	0.0	3.9	1.1	0.8
	Qwen2.5-Coder-3B-Instruct	5.0	4.7	6.5	11.6	11.6	11.6	10.0	9.0	2.2	8.0
	Qwen2.5-Coder-1.5B-Instruct	3.8	1.1	1.3	4.7	1.2	0.0	3.3	6.4	4.6	2.9
	Qwen2.5-3B-Instruct	51.3	50.6	36.4	47.7	48.8	45.4	45.6	41.0	39.8	45.2
	Qwen2.5-1.5B-Instruct	28.8	50.6	36.4	29.1	48.8	34.9	43.3	26.9	43.2	38.0
	deepseek-coder-1.3b-instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DeepSeek-R1-Distill-Qwen-1.5B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	10.0	23.5	15.6	20.9	26.7	19.8	31.1	15.4	12.5	19.5
	Falcon3-1B-Instruct	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	41.3	38.8	51.9	31.4	50.0	54.7	47.8	47.4	50.0	45.9
	Yi-Coder-1.5B-Chat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3b-code-instruct-128k	2.5	1.2	1.3	3.5	2.3	0.0	1.1	5.1	1.1	2.0
	granite-3.0-3b-a800m-instruct	2.5	2.4	1.3	2.3	3.5	1.2	3.3	2.6	3.4	2.5
	granite-3.0-2b-instruct	15.0	15.3	15.6	19.8	12.8	11.6	20.0	12.8	4.5	14.2
	EXAONE-3.5-2.4B-Instruct	10.0	38.8	31.2	24.4	24.4	22.1	38.9	33.3	17.0	26.7
	internlm2_5-1_8b-chat	0.0	1.2	1.3	5.8	2.3	1.2	2.2	2.6	0.0	1.8
	stable-code-instruct-3b	1.3	1.2	1.3	2.3	1.2	0.0	4.4	2.6	0.0	1.6
≤9B	CodeLlama-7b-Instruct-hf	3.8	2.4	1.3	3.5	2.3	2.3	7.8	5.1	2.3	3.4
	Llama-3.1-8B-Instruct	45.0	55.3	63.6	55.8	52.3	57.0	57.8	53.8	47.7	54.3
	codegemma-1.1-7b-it	15.0	17.6	13.0	15.1	18.6	15.1	17.8	14.1	9.1	15.0
	gemma-2-9b-it	48.8	49.4	55.8	44.2	44.2	45.3	48.9	52.6	56.8	49.6
	Qwen2.5-Coder-7B-Instruct	48.8	57.6	62.3	54.7	62.8	50.0	51.1	53.8	55.7	55.2
	Qwen2.5-7B-Instruct	51.3	45.9	55.8	47.7	54.7	52.3	51.1	52.6	53.4	51.6
	Marco-o1	60.0	51.8	67.5	53.5	62.8	55.8	52.2	62.8	56.8	58.1
	deepseek-coder-7b-instruct-v1.5	6.3	8.2	9.1	16.3	11.6	12.8	7.8	17.9	12.5	11.4
	deepseek-llm-7b-chat	0.0	2.4	3.9	7.0	7.0	5.8	5.6	11.5	4.5	5.3
	DeepSeek-R1-Distill-Qwen-7B	0.0	0.0	0.0	1.2	0.0	0.0	0.0	0.0	0.0	0.1
	DeepSeek-R1-Distill-Llama-8B	5.0	4.7	5.2	8.1	5.8	3.5	10.0	9.0	10.2	6.8
	Falcon3-7B-Instruct	42.5	41.2	27.3	41.9	37.2	29.1	34.4	24.4	39.8	35.3
	Baichuan2-7B-Chat	0.0	1.2	0.0	4.7	0.0	1.2	3.3	1.3	2.3	1.5
	Yi-Coder-9B-Chat	32.5	40.0	33.8	37.2	26.7	31.4	27.8	37.2	30.7	33.0
	Yi-1.5-9B-Chat	62.5	55.3	59.7	55.8	55.8	66.3	57.8	56.4	60.2	58.9
	granite-8b-code-instruct-128k	8.8	12.9	15.6	14.0	9.3	19.8	20.0	26.9	14.8	15.8
	granite-3.0-8b-instruct	20.0	25.9	28.6	31.4	31.4	19.8	26.7	19.2	25.0	25.3
	EXAONE-3.5-7.8B-Instruct	57.5	54.1	59.7	55.8	59.3	58.1	57.8	52.6	58.0	57.0
≤16B	CodeLlama-13b-Instruct-hf	7.5	7.1	9.1	10.5	5.8	11.6	13.3	10.3	14.8	10.0
	Qwen2.5-Coder-14B-Instruct	55.0	45.9	59.7	51.2	65.1	55.8	56.7	53.8	62.5	56.2
	Qwen2.5-14B-Instruct	83.8	84.7	83.1	80.2	82.6	84.9	81.1	73.1	84.1	81.9
	DeepSeek-Coder-V2-Lite-Instruct	15.0	10.6	5.2	15.1	15.1	15.1	10.0	19.2	10.2	12.8
	DeepSeek-V2-Lite-Chat	28.8	23.5	15.6	14.0	23.3	18.6	20.0	21.8	19.3	20.5
	DeepSeek-R1-Distill-Qwen-14B	66.3	63.5	76.6	65.1	67.4	70.9	73.3	45.0	72.7	66.8
	falcon-11B	12.5	5.9	3.9	8.1	3.5	9.3	8.9	14.1	6.8	8.1
	Falcon3-10B-Instruct	30.0	23.5	18.2	15.1	25.6	7.0	20.0	11.5	11.4	18.0
	Baichuan2-13B-Chat	0.0	0.0	1.3	1.2	0.0	1.2	1.1	1.3	0.0	0.7
	WizardLM-13B-V1.2	6.3	3.5	0.0	10.5	1.2	5.8	7.8	3.8	2.3	4.6
	starcoder2-15b-instruct-v0.1	15.0	12.9	20.8	14.0	8.1	15.1	17.8	20.5	19.3	15.9
	Mistral-Nemo-Instruct-2407	53.8	52.9	55.8	50.0	58.1	48.8	48.9	48.7	58.0	52.8
≤34B	CodeLlama-34b-Instruct-hf	6.3	10.6	14.3	10.5	8.1	12.8	5.6	14.1	15.9	10.9
	gemma-2-27b-it	67.5	68.2	64.9	67.4	69.8	64.0	67.8	61.5	60.2	65.7
	Qwen2.5-Coder-32B-Instruct	87.5	80.0	90.9	89.5	93.0	86.0	82.2	78.2	84.1	85.7
	Qwen2.5-32B-Instruct	90.0	89.4	96.1	90.7	86.0	86.0	92.2	85.9	88.6	89.5
	QwQ-32B-Preview	73.8	72.9	83.1	67.4	74.4	77.9	81.1	69.2	76.1	75.1
	Sky-T1-32B-Preview	78.8	82.4	90.9	76.7	80.2	81.4	85.6	76.9	84.1	81.9
	deepseek-coder-33b-instruct	5.0	4.7	2.6	12.8	3.5	8.1	13.3	19.2	11.4	9.0
≤72B	CodeLlama-70b-Instruct-hf	22.5	35.3	35.1	26.7	33.7	41.9	30.0	32.1	37.5	32.7
	Llama-3.1-70B-Instruct	83.8	67.1	75.3	74.4	80.2	81.4	81.1	65.4	81.8	76.7
	Llama-3.3-70B-Instruct	78.8	71.8	80.5	79.1	75.6	81.4	84.4	67.9	83.0	78.0
	Qwen2.5-72B-Instruct	92.5	94.1	89.6	89.5	90.7	93.0	88.9	89.7	89.8	90.9
	deepseek-llm-67b-chat	48.8	34.1	41.6	52.3	46.5	50.0	54.4	48.7	46.6	47.0
	DeepSeek-R1-Distill-Llama-70B	58.8	64.7	70.1	67.4	73.3	75.6	74.4	64.1	70.5	68.8
	WizardLM-70B-V1.0	53.8	41.2	48.1	50.0	53.5	55.8	47.8	50.0	40.9	49.0
	K2-Chat	63.8	58.8	74.0	61.6	64.0	60.5	64.4	60.3	47.7	61.7
	falcon-40b-instruct	1.3	4.7	5.2	5.8	2.3	2.3	7.8	6.4	2.3	4.2

Best Score Overall, Best Score within Scale

Table 12: Solution Identification - Hard (Invariant Accuracy).